UNIVERSIDADE ESTADUAL DO TOCANTINS

CAMPUS DE PALMAS

CURSO DE SISTEMAS DE INFORMAÇÃO

IMPLANTAÇÃO DE CLUSTER SGBD COM MICROSSERVIÇOS EM AMBIENTE DE PRODUÇÃO

DERRECK DANNILLO VAZ MARQUES

Palmas – TO

2017

UNIVERSIDADE ESTADUAL DO TOCANTINS
CAMPUS DE PALMAS
CURSO DE SISTEMAS DE INFORMAÇÃO

IMPLANTAÇÃO DE CLUSTER SGBD COM MICROSSERVIÇOS EM AMBIENTE DE PRODUÇÃO

DERRECK DANNILLO VAZ MARQUES

Trabalho de Conclusão do Curso apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação, sob a orientação do prof. Me. Douglas Chagas da Silva.

Palmas, Dezembro de 2017.

UNIVERSIDADE ESTADUAL DO TOCANTINS

CAMPUS DE PALMAS

CURSO DE SISTEMAS DE INFORMAÇÃO

IMPLANTAÇÃO DE CLUSTER SGBD COM MICROSSERVIÇOS EM AMBIENTE DE PRODUÇÃO

DERRECK DANNILLO VAZ MARQUES

Trabalho de Conclusão do Curso apresentado ao Curso de Sistemas de Informação da Universidade Estadual do Tocantins - UNITINS como parte dos requisitos para a obtenção do grau de Bacharel em Sistemas de Informação, sob a orientação do prof. Me. Douglas Chagas da Silva.

COMISSÃO EXAMINADORA

_____

Prof. Me. Douglas Chagas.

Orientador

_____

Professor

Convidado 1

_____

Professor

Convidado 2

Dedicatória

"Dedico esse trabalho ao meu pai Antonio Walter
e minha mãe Rita de Cássia."

Agradecimentos

Agradeço a minha familia por todo apoio durante esta jornada.

Agradeço aos meus amigos que me ajudaram com boas conversas e momentos descontraídos.

Aos colegas de trabalho, um especial agradecimento por seus ensinamentos e pelo direcionamento durante este trabalho.

Aos professores da instituição que auxiliaram em minha formação, principalmente meu orientador Me. Douglas Chagas, um grande obrigado.

"O homem tem o tamanho daquilo que se atreve a fazer."

— Jorge A. Livraga

Resumo

Novas tecnologias estão sempre surgindo no cenário da tecnologia de informação com o intuito de melhorar a performance e o gereciamento de recursos dos serviços providos. Porém, essas tecnologias precisam ser testadas antes de serem implantadas em produção. O *Docker*, mecanismo de contêiner, e o modelo de microsserviços são as tecnologias/modelos abordadas neste trabalho para a implantação de um *cluster* SGBD. Diante disso, este trabalho consiste em implantar um cenário com essas ferramentas, comentando as vantagens e desvantagens das mesmas. Por mais, este trabalho conta ainda com a utilização de uma ferramenta de *benchmark* para testar seus desempenhos, comparando os resultados obtidos nos testes, e mostrando se há viabilidade dessas ferramentas para produção.

Palavras-chave: *cluster* SGBD; Microsserviços; *Docker*

Abstract

New technologies are always emerging in the information technology scenario with the purpose of improving the performance and the management of resources of the services provided. But these technologies need to be tested before they are deployed in production. The Docker, container mechanism, and microservice model are the technologies/models discussed in this work for the implementation of a cluster DBMS. In view of this, this work consists in implanting a scenario with tools to comment the advantages and disadvantages of the same. Moreover, this work also has the use of a benchmark's tool to test their performances, comparing the results obtained in the tests and showing if these tools are viable for production.

Keywords: *cluster* SGBD; Microservices; *Docker*

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

No modelo de infraestrutura atual de Tecnologia da Informação (TI) faz-se necessário a integração de diferentes soluções e tecnologias para provimento de serviços. As instituições (privadas e públicas) cada vez mais precisam entregar serviços em curto período, e com uma heterogeneidade de tecnologias cada vez maior.

Neste contexto, aspectos como criação, configuração e gerência de serviço são de grande importância. Quanto maior a facilidade em desenvolver esses aspectos, melhor será o aproveitamento de recursos operacionais, de tempo e monetários, consequentemente melhor será a entrega do produto para o usuário final.

Desta forma prover serviços que propicie esses aspectos de maneira ágil e fácil, é a chave para um desempenho otimizado da infraestrutura das instituições.

Evidencia-se que as soluções de TI, de maneira geral, precisam oferecer características e funcionalidades que contribuam para a integração de diferentes plataformas, e ao mesmo tempo, forneça camadas específicas de isolamento, de forma a manter sistemas legados em funcionamento, garantindo a estabilidade necessária para manutenção desses serviços, visto que dependendo do porte da aplicação e/ou sistema a migração nem sempre torna-se uma tarefa fácil e rápida.

A utilização de contêineres promove muitas das facilidades mencionadas anteriormente. Neste contexto, conceitos como microsserviços apontam caminhos interessantes para entrega de serviços de maneira ágil e escalável. Desta forma, o estudo das possibilidades e desafios desta tendência de mercado passam a ser essenciais no processo de entrega de serviços e implantações de soluções em infraestrutura de TI.

Este trabalho objetiva realizar a implantação de um *cluster* para provimento de disponibilidade e escalabilidade de um Sistema de Gerenciamento de Banco de Dados (SGBD), fazendo uso do conceito de microsserviços a partir da implementação de contêineres para um ambiente de universidade pública. Além disso, visa-se ainda demonstrar os passos necessários para a implantação da solução, a partir de cenários de testes para representação de ambientes de homologação e produção.

O trabalho está organizado da seguinte forma: no capítulo 1 é abordado a parte introdutória, mostrando a proposta do trabalho; no capítulo 2 é apresentado um breve referencial teórico, apontando as principais tecnologias e conceitos envolvidos no processo de desenvolvimento do trabalho proposto; no capítulo 3 é apresentado a metodologia utilizada para resolução do problema apresentado neste trabalho, além de aspectos de como os testes foram realizados; por fim, nos capítulos 4 e 5 apresenta-se os resultados e considerações finais, respectivamente.

1.1 Justificativa

O uso dos recursos providos pelos contêineres é feito de maneira otimizada, quando comparada com modelo tradicional de virtualização monolítica (MOUAT, 2015), pois além de usar a API (Interface de Programação de Aplicativos) do kernel do SO diretamente, não necessita de um camada de virtualização. Desta forma, ao optar pelo uso de *contêineres*, a instância do serviço a ser implantado fará uso apenas dos recursos necessários para sua execução.

No cenário de TI, novas tecnologias estão sempre aparecendo com o intuito de melhorar o desempenho de serviços e também a utilização otimizada de recursos, porém essas tecnologias devem ser testadas antes de ir para produção, pois as mesmas podem apresentar problemas ou desempenho inferior. Desse modo, esse trabalho irá testar duas novas tecnologias/modelo, mecanismo de contêiner Docker e microsserviços, para provimento de um serviço em *cluster*.

Assim, o estudo de ferramentas próprias para criação e gerência de *cluster* com microsserviços é fundamental para avaliação da viabilidade de implantação em ambientes de produção, garantindo a escalabilidade e disponibilidade desses serviços.

1.2 Problema

No modelo tradicional a implementação de um cluster, para provimento de alta disponibilidade, requer a utilização de inúmeros protocolos, tais como: DRBD, Heartbeat e Mon, dentre outros (PEREIRA, 2005). Essas tecnologias precisam trabalhar de forma integrada, o que de maneira geral, configura-se como uma tarefa árdua, visto a necessidade de um domínio de todas as ferramentas envolvidas, tornando o escalonamento do serviço uma tarefa ainda mais complexa.

A utilização de máquinas virtuais (VM) e/ou servidores dedicados para entrega de serviços, impõem um modelo na qual necessita-se de virtualizador, isto é, uma camada adicional de software, que por conseguinte requer mais recursos de hardware, e que não necessariamente contribua para um maior desempenho da aplicação ou do serviço implantado. Este modelo embora bastante funcional, é caro computacionalmente (TANENBAUM, 2009).

Assim, o modelo baseado em microsserviços apresenta-se como uma solução simples e eficaz para a implementação de mecanismos de *cluster* e alta disponibilidade para serviços em ambientes de produção, sem a imposição da necessidade do aumento do parque computacional, ou ampliação dos recursos de hardware dos servidores em uso na instituição.

1.3 Objetivos

1.3.1 Objetivo Geral

- Realizar a implantação de um *cluster* SGBD utilizando microsserviços, no contexto de uma universidade pública.

1.3.2 Objetivos Específicos

- Analisar as vantagens e desvantagens da utilização de microsserviços;
- Testar um ambiente com microsserviços de modo a validar o estudo realizado;
- Compreender os aspectos de paravirtualização e isolamento de aplicações;
- Estudar os conceitos de utilização de microsserviços no contexto de infraestrutura ágil;
- Medir a eficiência técnica da solução proposta para ambientes de produção.

2 Referencial Teórico

2.1 Cluster

Um *cluster* é a composição de vários CPU que trabalham em conjunto para resolver um trabalho computacional. Isso implica em várias máquinas trabalhando em grupo numa mesma rede local (SILBERSCHATZ; GALVIN; GAGNE, 2010). A seguir, a Figura 1 demonstra os componentes do *cluster* e exemplifica o conceito citado anteriormente.

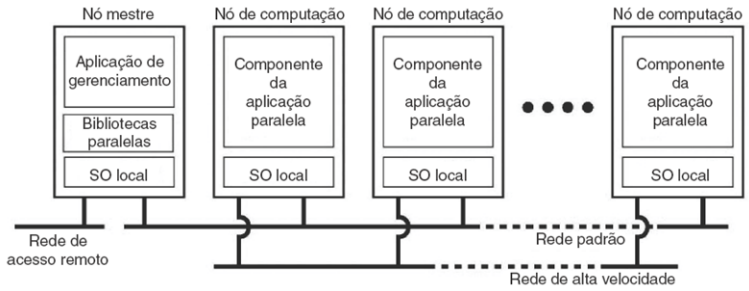

Figura 1 – Composição de Cluster fonte: (TANENBAUM; STEEN, 2007)

O funcionamento de um *cluster*, do ponto de vista computacional, ocorre quando um serviço é executado em vários computadores ou estações de trabalho de forma paralela. A comunicação do cluster é feita através de uma rede local (TANENBAUM; STEEN, 2007).

A computação em *cluster* é muito utilizada no provimento de alto desempenho e no uso otimizado de recursos computacionais. Para isso, os integrantes do *cluster* estão de alguma forma associados e trabalhando em conjunto (SILBERSCHATZ; GALVIN; GAGNE, 2010).

Na Figura 2 é exemplificada as camadas de *cluster* de acordo com (TANENBAUM; STEEN, 2007).

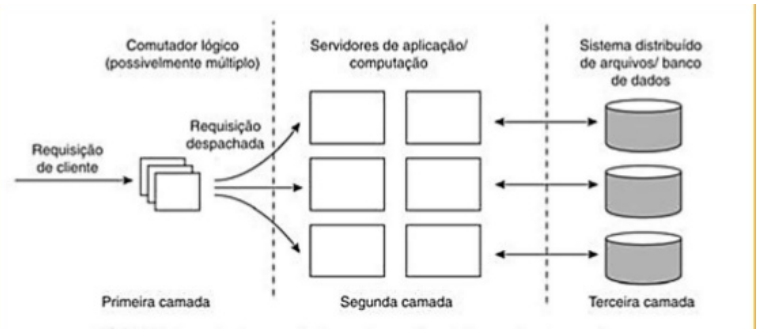

Figura 2 – Camadas de um Cluster fonte: (TANENBAUM; STEEN, 2007)

Observa-se ainda na Figura 2, a camada de tratamento de requisições dos clientes; o grupo de servidores que provê o serviço; e a camada de sistema de arquivo distribuído utilizado pelos servidores.

Na criação de um sistema de alta disponibilidade e desempenho é utilizado computação em *cluster*. Essas características são obtidas adicionando algum tipo de redundância, podendo ser de hardware ou software (SILBERSCHATZ; GALVIN; GAGNE, 2010).

Para o funcionamento correto do *cluster* é necessário a existência de uma camada de software, que irá trabalhar no gerenciamento de como os nós irão executar os serviços e também monitorar as falhas e fazer suas correções (SILBERSCHATZ; GALVIN; GAGNE, 2010).

Existem três tipos de cluster (DEITEL; DEITEL; CHOFFNES, 2005):

- *Cluster* de alto desempenho: utiliza-se de todos os recursos computacionais de um grupo de computadores, conseguindo assim grande capacidade de processamento;
- *Cluster* de alta disponibilidade: há servidores de produção executando os serviços e há servidores reservas que assumem a responsabilidade do serviço, caso ocorra alguma falha nos servidores de produção;
- *Cluster* de balanceamento de carga: um nodo é escolhido para tratar as requisições e encaminhar para os outros nodos do *cluster*.

2.2 Microsserviços

O conceito de um sistema implementado no modelo de microsserviço refere-se à constituição de serviços independentes trabalhando em conjunto. Dessa forma, é possível utilizar diferentes tecnologias para provê os serviços (NEWMAN, 2015).

Assim, microsserviços referem-se a um conjunto de serviços, que executam-se de forma independente e comunicam-se através de uma rede. Uma aplicação é construída com cada microsserviço oferecendo uma funcionalidade (JARAMILLO, 2016).

2.2.1 Heterogeneidade de tecnologia

Em virtude da grande diversidade de tecnologias no mercado, com propósitos e soluções semelhantes, mas funcionando melhor em cenários específicos, conseguir utilizar diferentes tecnologias, traz flexibilidade para implantação de serviços em uma empresa ou instituição. Para cada cenário em uma empresa ou instituição, onde se faz necessário ter maior performace , é essencial trabalhar com tecnologias diferentes, gerando soluções de serviços plausíveis, com boa escalabilidade e manuntenção no decorrer da aplicação.

Por exemplo, no modelo monolítico existe um grande risco com a implantação de novas tecnologias em um sistema em produção, pois o sistema está fortemente acoplado. Caso ocorra uma falha, todo o sistema pode parar de funcionar, sendo uma falha difícil de tratar, logo, tendo-se em conta a dificuldade em isolar cada módulo da aplicação e em encontrar a respectiva falha. Assim por ter grande flexibilidade no tipo de tecnologia utilizada no modelo de microsserviços,

ocorre a oportunidade de testar novas tecnologias com grande controle de falhas (NEWMAN, 2015).

2.2.2 Resiliência

Ter um sistema tolerante às falhas é de grande importância para que o usuário tenha uma boa experiência de uso e aumente a confiabilidade do sistema. A partir da utilização do modelo de microsserviços, é possível configurar o sistema para tratar as falhas.

Por serem serviços isolados, a identificação do problema pode ser feita rapidamente. Após tirar o serviço falho de operação, cria-se um novo serviço com a mesma funcionalidade para substituição . Um outro meio de tratar as falhas pode ser criado no planejamento do sistema, no qual é implantando mecanismos de redundância (NEWMAN, 2015).

2.2.3 Implantação de Serviços

Adicionar novas funcionalidades a uma aplicação é essencial para tratar de problemas e para melhorar uma aplicação em produção. Quando se tem o processo de atualizações, existe um grande risco de implantação, visto que elas modificam fluxo de uma aplicação. Assim, conseguir fazer modificações sem grandes problemas é importante em ambientes de mudanças constantes.

Implantar novas atualizações no modelo monolítico pode gerar problemas ao funcionamento da aplicação, pois, com a forte conexão dos módulos, a mudança em parte do sistema pode resultar em falha completa do mesmo. Como os serviços implementados, no modelo de microsserviços, não estão fortemente acoplados, proporciona-se, assim, menos riscos para novas atualizações, sendo possível manter as entregas contínuas do serviço.

Por não serem tão conectados, serviços implementados com microsserviços, proporciona menos riscos para novas atualizações, e assim é possível manter entregas contínuas do serviço (NEWMAN, 2015).

2.2.4 Alinhamento Organizacional

Conseguir manter o foco de uma equipe em um serviço é essencial para qualidade final de uma entrega. Utilizando o modelo de microsserviços, é possível organizar melhor as equipes pelo desenvolvimento de um serviço.

Devido o fato de uma equipe está se concentrando somente em um único serviço, outros problemas não afetarão seu trabalho, logo o fluxo de atividades irá acontecer sem intervenções, e também a qualidade do serviço irá melhorar (NEWMAN, 2015).

A utilização de microsserviços para entrega rápida de produtos, como aplicações *web*, é feita com a divisão da aplicação em partes menores, permitindo delegar partes da aplicação para diferentes grupos na instituição. Dessa forma, não há muita dependência de um grupo com outro,

porque cada grupo pode desenvolver sua parte de forma separada, focando-se mais na entrega do módulo para a aplicação final (JARAMILLO, 2016).

2.2.5 Escalonamento de Serviços

Em tempos de grande demanda, uma aplicação pode apresentar problemas em tratar muitas requisições. Desse modo, é necessário escalar as funcionalidades do serviços, e por conseguinte, entender como o escalonamento torna-se essencial quando aplicado ao modelo de microsserviços.

Em comparação ao sistema monolítico, quando ocorre a necessidade de escalonar algum serviço, toda aplicação é escalonada. Por ser fortemente acoplada, não é possível escalonar por funcionalidade, em vez disso, escalona-se toda a aplicação. Isso traz algumas perdas de recurso, pois, quando o sistema precisa de escalonamento, nem sempre todas as funcionalidades necessitam serem escalonadas. A partir da divisão em módulos, com microsserviços é possível escalonar de maneira otimizada, já que é possível escalonar por módulos ao invés de toda a aplicação (NEWMAN, 2015).

No modelo tradicional, por não dividir o módulo, a escalabilidade não é feita da melhor maneira, uma vez que é criada uma nova instância de todo o módulo. Assim, a melhor maneira, neste caso, seria escalonar somente a funcionalidade que necessita de recursos. O escalonamento por funcionalidade é possível com os microsserviços, porque seus componentes são independentes (JARAMILLO, 2016).

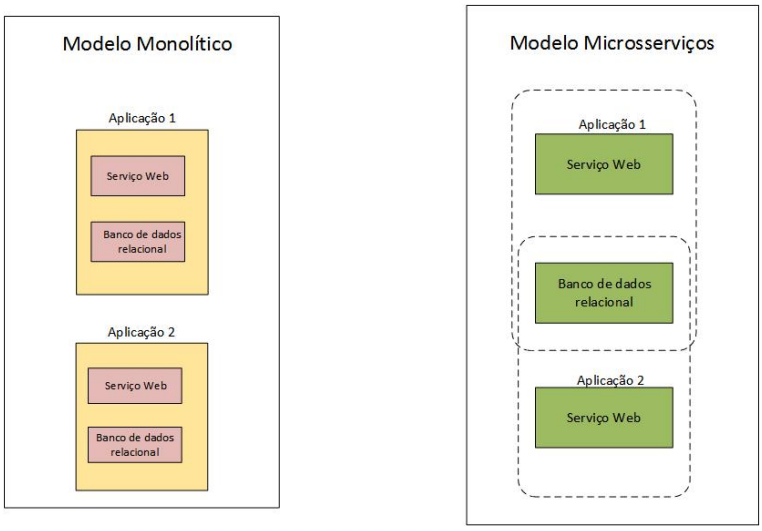

Figura 3 – Modelo Monolítico e microsserviços fonte: (POSADAS, 2017)

Na Figura 3, é possível entender melhor como os microsserviços se diferenciam do modelo tradicional. Na parte da esquerda (modelo tradicional), a aplicação tem todas as funcionalidades juntas. Porém há alguns gargalos nesse modelo, como, por exemplo, se uma

funcionalidade tiver um erro, será necessário vários procedimentos para substituir o módulo que falhou e a aplicação será comprometida com a falha (JARAMILLO, 2016).

O escalonamento é feito com todas as funcionalidades no modelo tradicional. Desta forma, alguns módulos que não estão sendo muito utilizados acabam ganhando mais recursos sem necessidade. A modificação de uma parte da aplicação impõe a configuração de todas as partes para o funcionamento correto mediante nova atualização (NEWMAN, 2015).

Nota-se ainda na Figura 3, na parte da direita, existe a divisão do serviço em funcionalidades. Essa divisão permite otimizar o uso da aplicação em comparação ao modelo tradicional. No escalonamento será aumentado apenas a funcionalidade que se fizer necessária. Desta forma, um erro pode ser reparado com a substituição do módulo que falhou.

2.2.6 Otimização para Substituição

A utilização de sistemas legados e aplicações novas é muito comum em ambientes de produção nas diversas instituições . Esses sistemas legados dificilmente são substituídos, porque existe um grande custo e risco na sua substituição. Contudo, A utilização de microsserviços, esse problema é resolvido, pois a substituição não traria tantos riscos ao sistema principal (NEWMAN, 2015).

Conforme mencionado anteriormente, ao invés de ter um grupo de serviços em conjunto, tem-se a fragmentação do conjunto de serviços em pequenos serviços. Essa divisão fornece uma maior liberdade para atualizar, escalar e modificar a aplicação. Enfim, é possível desenvolver e implantar uma infraestrutura ágio, tendo em vista as características dos microsserviços, assegurando, com isso, a entrega dos produtos de maneira otimizada para o usuário final (NEWMAN, 2015).

2.3 Virtualização

A virtualização consiste em uma técnica que simula um *hardware* ou um sistema operacional (SO). Desta forma, é possível executar vários serviços em uma mesma máquina. Para a execução dos serviços, é criada uma base, podendo ser uma base de *hardware* e SO virtual, para implementação de tudo que o serviço necessita para ser executado (TANENBAUM, 2009).

Algumas vantagens de se utilizar a virtualização (TANENBAUM, 2009):

1. Executar múltiplas instâncias de sistemas operacionais, logo, sendo possível otimizar a utilização dos serviços e recursos;
2. Isolamento de recursos entre os ambientes virtuais, pois caso ocorra, uma falha em um ambiente virtual, os outros ambientes virtuais não serão afetadas ;
3. Economia de dinheiro com compras de *hardware*, de eletricidade e espaço no *data center*.

A virtualização permite a criação de vários ambientes lógicos, que são aproveitados por administradores de uma infraestrutura para o provimento de serviços, sendo que cada serviço pode ser implantado em um ambiente virtual diferente. Esse modelo prover uma maturidade técnica e molda basicamente toda a infraestrutura de uma empresa de médio e grande porte.

2.3.1 Máquinas Virtuais

Para implantação de serviços com máquinas virtuais(VM do acrônimo em inglês virtual machine) utiliza-se o conceito de virtualização completa. Basicamente ,para o funcionamento de máquinas virtuais, é necessário criar um ambiente virtual com tudo que for necessário para execução de um sistema operacional (SILBERSCHATZ; GALVIN; GAGNE, 2010).

2.3.1.1 Características e Funcionalidades

A parte principal do sistema virtualizado é o *hypevisor*, que é instalado diretamente sobre *hardware*, viabilizando a execução de várias máquinas virtuais. Uma determinada máquina virtual, pode inclusive, criar seu próprio *hardware* (também virtual) e ter diferentes sistemas operacionais rodando em uma mesma máquina física (TANENBAUM, 2009).

Em virtude dos ambientes serem isolados entre si, é necessário utilizar de técnicas compartilhamento de arquivos. Conforme (SILBERSCHATZ; GALVIN; GAGNE, 2010) é sugerido o compartilhamento de arquivos entre os ambientes através de volume de dados compartilhados, ou ,ainda, a implementação de uma rede para troca de dados.

A capacidade de salvar o estado de máquina virtual pode ser utilizada na recuperação de um erro ou retorno de uma atualização que não ocorreu adequadamente. Além disso, ela pode ser utilizada também para migração das máquinas virtuais entre servidores para o balanceamento de carga(TANENBAUM, 2009).

A utilização de máquinas virtuais é uma técnica utilizada há muito tempo. As primeiras máquinas virtuais utilizadas comercialmente foram as da IBM, onde um mainframe disponibilizava várias máquinas virtuais, cada uma disponibilizando seu próprio sistema operacional (SILBERSCHATZ; GALVIN; GAGNE, 2010). O uso de máquinas virtuais para provê serviços é um modelo já estabilizado no mercado, porém, com novas tecnologias surgindo, alguns problemas desse modelo ficam evidentes.

Os maiores problemas da implantação de serviços com VM são as quantidades de recursos computacionais empregados, e a utilização do modelo monolítico na organização do serviço

No tópico a seguir serão apresentados os conceitos para o provimento de serviços baseados em contêineres , no qual serão demonstrados o modelo de implantação de serviços que atende às novas tecnologias e os problemas encontrados na utilização de VMs.

2.3.2 Contêineres

O tipo de virtualização utilizada por contêineres é a paravirtualização. Nessa virtualização, é possível aproveitar melhor os recursos, e conseguir os mesmos resultados de VMs.

Ao invés de se criar um ambiente virtual, é oferecido para o ambiente convidado um sistema semelhante ao do hospedeiro. O funcionamento acontece com a modificação do sistema operacional hóspede, para que este não precise de um virtualizador. O convidado deve modificar sua base para aproveitar os recursos disponibilizados, já que não é criado um ambiente completo e sim o compartilhamento do sistema hospedeiro (SILBERSCHATZ; GALVIN; GAGNE, 2010).

Para utilizar esse tipo de virtualização, é preciso ter um meio de comunicação do sistema operacional hóspede com os recursos do hospedeiro. Por conseguinte, o ambiente será executado como um processo dentro sistema operacional hospedeiro (TANENBAUM, 2009).

As vantagens desse tipo especial virtualização, de acordo com (SILBERSCHATZ; GALVIN; GAGNE, 2010), está na utilização eficiente dos recursos, além desta possuir uma camada menor de virtualização, trazendo otimização do uso do *hardware*.

2.3.2.1 Características e Funcionalidades

Para agrupar tudo que uma aplicação precisa para ser executada, os contêineres disponibilizam meios para a configuração de um ambiente encapsulado, que proverá tudo que for necessário para execução da aplicação (ZHANG; K., 2016). Os contêineres utilizam a virtualização, especificamente a paravirtualização, porque, a partir dela, o sistema operacional é virtualizado e a aplicação é executada sem a necessidade de virtualização do *hardware* (SILBERSCHATZ; GALVIN; GAGNE, 2010).

Para o *host* [1] hospedeiro, os contêineres são apenas processos, e há um grande controle para executar e parar os mesmos (SILBERSCHATZ; GALVIN; GAGNE, 2010). Por serem apenas processos para o sistema operacional, há um controle maior das aplicações que estão sendo executadas. É possível ter várias aplicações rodando em uma mesma máquina, isoladas e se comunicando através de rede local. É possível também limitar o que os contêineres podem utilizar, como quantidade memória, CPU e espaço em disco (SALAH M. JAMAL ZEMERLY, 2017).

É possível ainda, ter maior consistência de uma infraestrutura com a portabilidade dos contêineres, porque o que for produzido em teste pode ser implantado igualmente em um ambiente de produção. A mesma coisa acontece com o recurso computacional, pois, independentemente de ser em um notebook ou servidor, a aplicação será a mesma (JARAMILLO, 2016).

[1] Qualquer máquina ou computador que esteja conectado a uma rede

Os conceitos de microsserviços são facilmente implementados em contêineres, uma vez que é possível separar uma aplicação em pequenos módulos. Neste caso, cada contêiner será encarregado de uma funcionalidade . A comunicação dos módulos implantados em contêineres pode ser feita a partir de uma rede interna, facilitando o funcionamento de uma aplicação.

2.4 Docker

Para gerenciar as aplicações em contêineres, avaliou-se a ferramenta *Docker* [2]. O *Docker* provê meios para a criação de tudo que é necessário para o funcionamento de contêiner, como a isolação de um espaço na memória e o controle de quantidade de recursos usados.

O Docker consiste em uma Ferramenta de código aberto que disponibiliza meios para criação, execução, teste e implantação de aplicações em *contêiner* (SALAH M. JAMAL ZEMERLY, 2017). A aplicação que compõem os serviços requerem um ambiente com todas as dependências. Tal ambiente funciona em modo de paravirtualização, e é disponibilizado pelo *Docker* (JARAMILLO, 2016).

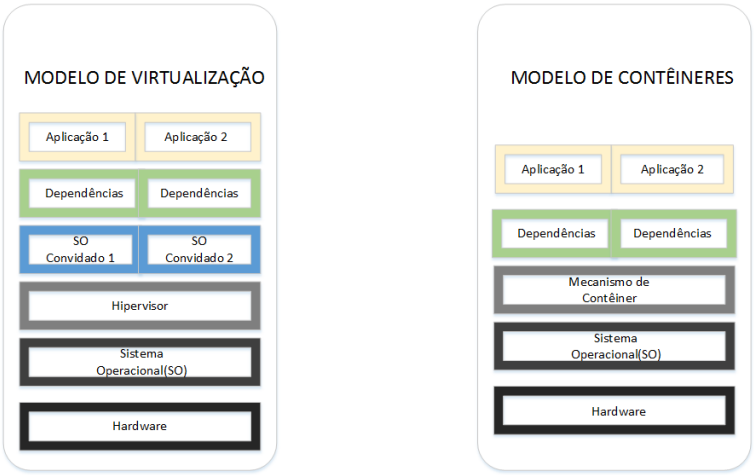

Figura 4 – Comparação virtualização completa e paravirtualização fonte: Adaptado (SALAH M. JAMAL ZEMERLY, 2017)

A engine *Docker* implanta todas as vantagens do uso de *contêiner* citadas no tópico anterior: portabilidade, encapsulamento e otimização do uso dos recursos. Nota-se que, na Figura 4, na parte esquerda, onde se encontra a virtualização completa, a aplicação estará sendo executada em um sistema operacional, que por sua vez precisará de um *hipervisor* para a virtualização de um hardware. Assim, serão percebida quais camadas de software são necessárias para a execução do serviço.

Detalha-se ainda na Figura 4, na parte direita da imagem, o processo de paravirtualização. A aplicação estará sendo executada em um sistema operacional virtualizado, que irá fazer

[2] Disponível em: https://www.docker.com

chamadas diretas no *hardware* do sistema operacional hospedeiro (as chamadas diretas serão feitas com funcionalidades do *Docker*).

O isolamento dos serviços é necessário para que não ocorram problemas no momento do acesso de recursos, e também para que, quando um serviço falhar, não afete diretamente os outros serviços. O *Docker* oferece meios de isolamento próprios. Caso um ambiente tenha vários serviços sendo executados, é importante provê meios de isolar esses serviços, pois isso possibilita amenizar ou identificar um problema. Um dos intuitos desse processo isolamento é que , caso ocorra algum problema, ele afetará apenas um contêiner e não todas as partes da máquina, principalmente onde o *Docker* está instalado (NAIK, 2016).

A utilização de arquivos descrevendo as aplicações, possibilita ainda executar um mesmo software em outra máquina ou servidor, o que gera a diminuição de tempo com a configuração da aplicação e com a portabilidade entre os ambientes. Por ter pequenas aplicações, a implantação, descoberta e correção de erro são facilitadas (KANG MICHAEL LE, 2016).

2.5 Ferramentas para Implantação de *Cluster* Linux

2.5.1 DRBD

O DRBD é uma solução para a replicação de dados entre volumes de servidores, com armazenamento replicado baseado em software. Desta maneira, é possível replicar os dados de um servidor para outro, e caso um servidor pare de funcionar, o servidor reserva assume os dados do servidor de produção (PERKOV NIKOLA PAVKOVIC, 2007). O funcionamento do DRBD é mostrado na Figura 5.

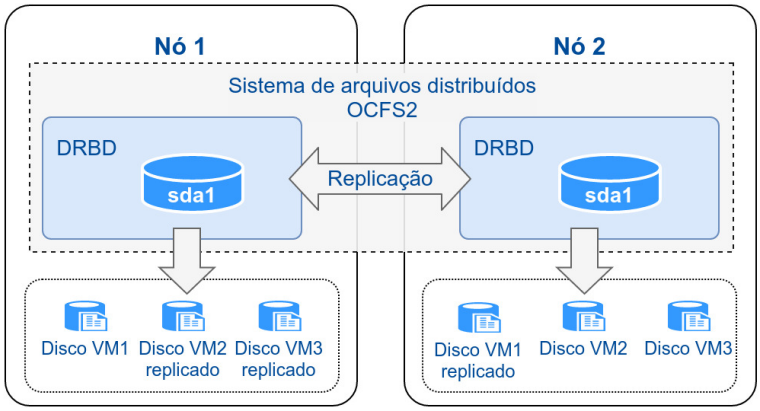

Figura 5 – Funcionamento do DRBD fonte: (EMER, 2016)

Há três pontos importantes a respeito do funcionamento do DRBD (RIASETIAWAN; ASHARI; ENDRAYANTO, 2015).

1. Replicação de Dados: a replicação de dados é momentânea após a gravação de dados. Isso implica em uma operação rápida de replicação de dados de um servidor para outro ;

2. Transparência da replicação: de forma transparente, a replicação de dados é feita para os clientes do DRBD , pois os mesmos não sabem que seus dados estão replicados em outras bases;

3. Comunicação: existem duas maneiras de informar que os dados já foram gravados em outros *hosts*: síncrona e assíncrona.

   a) A síncrona ocorre quando os dados são gravados em todos os *host*. A aplicação gera uma notificação informando que os dados foram replicados às aplicações.

   b) A assíncrona ocorre quando as aplicações são notificadas da gravação dos dados localmente.

O DRBD possui um conjunto de ferramentas para gerenciar a configuração e as funcionalidades. Essas ferramentas são: drbdadm, drbdsetup e drbdmeta [3] (RIASETIAWAN; ASHARI; ENDRAYANTO, 2015).

i) Drbdadm: serve para administração e configuração da aplicação através de arquivos, e funciona como uma interface para drbdsetup e drbdmeta.

ii) drbdsetup: serve para configuração do módulo DRBD, e todas as configurações do módulo precisam ser passadas por linha de comando.

iii) drbdmeta: é um utilitário para todas as operações referentes aos metadados da aplicação. O mesmo possibilita criar, modificar e remover metadados.

Alguns conceitos utilizados pelo DRBD (RIASETIAWAN; ASHARI; ENDRAYANTO, 2015):

Recursos: são referências para um grupo de dados replicados, possuindo: nome, volumes, um bloco virtual e um meio de comunicação para troca de dados replicados.

Volumes: os dados replicados estão em volumes separados, compartilhando um fluxo de dados replicados.

Bloco Virtual: cada recurso DRBD mantém vários blocos virtuais nos volumes associados.

2.5.2 Corosync

O Corosync[4] é uma ferramenta para a comunicação entre os servidores de um *cluster*. Ele trabalha em conjunto com outras ferramentas para provê uma alta disponibilidade (KHAN; TOEROE; KHENDEK, 2017).

O Coronsync possui recursos para fornecer a comunicação entre os servidores, que são (KHAN; TOEROE; KHENDEK, 2017):

[3] Disponível em: https://docs.linbit.com/
[4] Disponível em: http://corosync.github.io/corosync/

- Modelo de comunicação privada entre um grupo;
- Gerenciamento de disponibilidade: conforme sua configuração, o Corosync irá procurar e corrigir as falhas no *cluster*;
- Banco de dados: para armazenar configurações e dados para estatísticas.

2.5.3 Heartbeat

O Heartbeat[5] consiste em uma ferramenta para a associação e comunicação de servidores. O mesmo provê funcionalidades para a implantação de uma estrutura de *cluster*. A partir do cluster em funcionamento, a ferramenta também monitora os nós e trata das falhas do nodo de produção (ZAMINHANI, 2008).

O Heartbeat possui funcionalidades para gerenciar o *cluster*, sendo elas (ZAMINHANI, 2008):

- gerenciar um mecanismo de comunicação que permite a comunicação entre os nós do *cluster*;
- Consultas de configuração;
- A verificação da disponibilidade dos nodos: é feita através da comunicação e da coleta de informações a respeito da conectividade;

Na Figura 6 é mostrado como funciona a estrutura do Heartbeat, na qual há dois servidores que atendem às requisições dos clientes. Um dos servidores fica de reserva. Desse modo, assim que o Heartbeat capta uma falha no servidor principal, o servidor reserva assume.

2.5.4 Pacemaker

A ferramenta Pacemaker [6] é um gerenciador de recursos de uma arquitetura *cluster*. A sua funcionalidade para provê alta disponibilidade está na recuperação de falhas no sistema de uma arquitetura de *cluster*, já em funcionamento (PERKOV NIKOLA PAVKOVIC, 2007).

Quando o *cluster* é iniciado, um nodo é escolhido para ser o mestre do *cluster*. A partir desse nodo mestre, todas as configurações e serviços feitos são replicados para os outros nodos (BENZ; BOHNERT, 2015).

[5] Disponível em: http://linux-ha.org/wiki/Heartbeat
[6] Disponível em: https://www.clusterlabs.org/

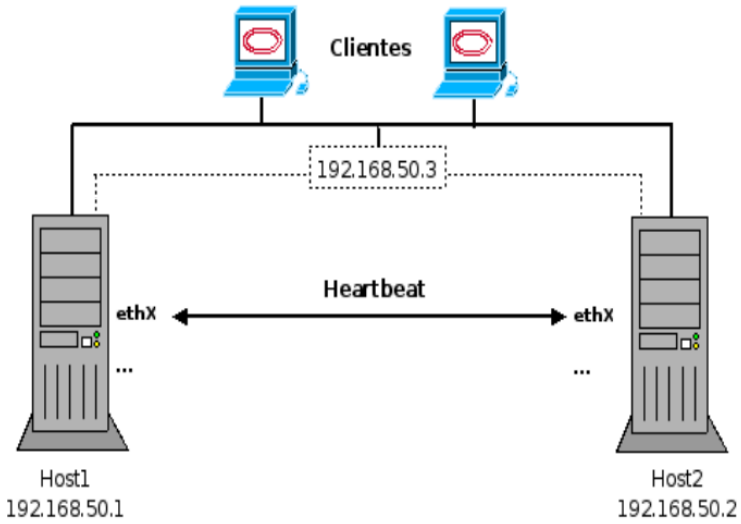

Figura 6 – Infraestrutura de Cluster com Heartbeat: Adptado (ZAMINHANI, 2008)

2.6 Ferramentas para Implantação de *Cluster* com Contêiner

Duas ferramentas foram avaliadas para o gerenciamento de *clusters* de contêineres, sendo elas: *Docker Swarm*[7] e *Kubernetes*[8]. Essas ferramentas são apresentadas nas próximas seções.

2.6.1 Docker Swarm

O *Docker Swarm* é uma ferramenta nativa do *Docker*, possuindo uma arquitetura composta por nodos gerenciadores e nodos normais. Assim, é possível tanto ter mais nodos para o gerenciamento quanto ter diversos nodos normais. Convenciona-se por nodos normais, os nodos cujas as características sejam: executar serviços no *cluster*; não conseguir manipular as funcionalidades e os recursos do *cluster*.

O uso de mais de um nodo para o gerenciamento, sugere que o *cluster* provê tolerância as falhas na parte de administração das requisições e recursos do *cluster*. Os nodos normais são encarregados da execução dos serviços, e os gerenciadores agendam e orquestram as atividades dentro do *cluster* (MOUAT, 2015).

A integração do *Docker Swarm* no *Docker engine* possibilita a criação de *cluster*, usando comandos dentro do *Docker*. Desta forma, é aproveitada toda a estrutura do *Docker* para gerir os serviços no *cluster*, como a utilização de imagens na criação de serviços no *cluster* (GROßMANN; KLUG, 2017).

Gerenciada pelo *Docker Swarm*, a quantidade de réplicas executadas por um serviço no *cluster*, estar sempre de acordo com a configuração pré-definida do serviço. E, caso algum serviço falhe, um novo serviço é executado, em substituição (MOUAT, 2015).

[7] Disponível em: https://docs.docker.com/engine/swarm
[8] Disponível em: https://kubernetes.io/

Existem dois tipos de nodo no *Docker Swarm*: o *gerenciador* e o *agente*. O gerenciador atua como controlador do *cluster*, e é o responsável por gerenciar a quantidade de réplicas e de carga para cada nodo. O *agente* executa o serviço provido pelo *cluster* (GROßMANN; KLUG, 2017).

A partir da possibilidade de se criar uma rede para o *cluster*, o *Docker Swarm* prepara automaticamente um endereço de rede para os *contêineres* que estão executando os serviços. Além disso, o *Docker Swarm* tem o serviço de DNS(domain name server) , encontrando e analisando os contêiners do *cluster* (NAIK, 2016).

Para melhor uso do *cluster*, um nodo não pode ficar sobrecarregado, assim como não pode ficar ocioso. Em ambos os casos, utiliza-se o balanceamento de carga. O balanceamento entre os nodos que compõem o *cluster* pode ser executado por ferramentas externas ou através da configuração do *Docker Swarm* (NAIK, 2016).

Na descoberta de serviço, um nodo com *Docker engine* inicia-se no modo *Docker Swarm*. A partir da inicialização no *Docker Swarm*, um *token* é gerado. Esse *token* é utilizado na adição de novos nós ao *cluster*, em conjunto com um comando específico (MOUAT, 2015).

2.6.2 Kubernetes

Outra ferramenta de criação e gerência de *cluster* é o *Kubernetes* [9], que também provê múltiplas funcionalidades para gestão de um *cluster*.

O *kubernetes* é uma ferramenta de código aberto do *Google*, que gerencia aplicações em *contêiner* e em vários *host* de um *cluster*. Ele tem como finalidade simplificar a utilização de aplicações, organizadas em microsserviços.

Umas das vantagens do uso de *Kubernetes*, é que seu desenvolvimento não se destina exclusivamente ao *Docker*, o que possibilita a troca de solução de *contêiner*, caso seja necessária (TSAI et al., 2017).

Para o funcionamento correto, o *Kubernetes* possui elementos com funções definidas para a execução do *cluster*, conforme detalhada a seguir (TSAI et al., 2017).

a) minios: são os *host* que estão associados ao *cluster*;

b) kubelet: aplicação do *Kurbenetes* que executa no *host*;

c) pods: são os *contêineres* que estão associados ao cluster;

d) replications controller: é o encarregado de controlar a quantidade de replicas de um serviço;

[9] https://kubernetes.io/

e) services: gera uma faixa de endereço de rede para o *replications controller*, para que o mesmo distribua automaticamente os endereços para os *pods*;

O *kubernetes* possibilita ainda escalonar um serviço para atender uma demanda tanto alta quanto baixa. Para isso, são criados mais contêineres ou paralisados os já existentes. Essa configuração pode ser feita através de uma interface de usuário ou de acordo com as configurações prévias para o uso da CPU [10] (GROßMANN; KLUG, 2017).

A respeito do controle de falha para atualização, as mudanças nas aplicações acontecem de forma gradativa, enquanto é verificado o estado dos contêineres, para que, caso ocorra falha, a alteração seja parada e a mudança possa ser desfeita (TSAI et al., 2017).

Quando contêineres não respondem à verificação do *Kubernetes*, eles são reiniciados ou substituídos, e os *contêineres* falhos não são divulgados para serem utilizados (TSAI et al., 2017).

[10] unidade central de processamento

3 Metodologia

A metodologia, proposta neste trabalho, tem como objetivo detalhar a definição do cenário de implantação do *cluster*, bem como dos testes realizados, a partir disso, verificar o seu funcionamento correto, simulando sua utilização pela comunidade interna de uma universidade.

A partir da implementação dos testes, foram realizadas as comparações entre um *cluster* SGBD com microsserviços e uma VM com o mesmo SGBD instalado, representando o modelo tradicional.

Para criar a estrutura do cenário *cluster*, avaliou-se duas ferramentas de gerenciamento de serviços, conforme destacado na seção 2.6, sendo elas: *Docker Swarm* e *Kubernetes*. As duas opções oferecem meios para balancear carga, para a descoberta de serviço e para atualização progressiva de serviço. Contudo, optou-se pela ferramenta *Docker Swarm*, pelas características apresentadas na seção 2.6.1, e também por ter uma boa documentação e ser nativa do *Docker*.

Na definição e criação do cenário de testes, utilizou-se as seguintes ferramentas/aplicações : Docker [11] , Docker Swarm [12], Oracle VirtualBox [13], MYSQL Server [14], MySQL Cluster [15], linguagem de programação Python [16] e Sysbench [17].

O papel de cada uma dessas ferramentas, é destacado a seguir:

- O Docker Swarm é a parte responsável por gerenciar o *cluster*. Ele também é responsável por adicionar novos *hosts*, além de distribuir a carga entre os nodos do *cluster*.
- O VirtualBox é o *hypervisor* utilizado para criação das VMs.
- O MySQL Server é a solução de banco de dados utilizada na VM para os testes de comparação.
- O MySQL Cluster foi a imagem utilizada para criação dos contêineres *Docker*.
- O Python é a linguagem de *scripts* utilizada para criar os testes de stress e coletar os dados nos cenários.
- O Sysbench é a ferramenta utilizada em conjunto com o Python. O mesmo serviu para gerar stress nos SGBDs.

Para o provimento do cenário de testes , com o objetivo de avaliar as ferramentas e as soluções estudadas, utilizou-se os materiais detalhados na Tabela 1.

[11] Disponível em: https://www.docker.com/
[12] Disponível em: https://docs.docker.com/engine/swarm/
[13] Disponível em: https://www.virtualbox.org/
[14] Disponível em:https://www.mysql.com/
[15] Disponível em:https://hub.docker.com/r/mysql/mysql-cluster/
[16] Disponível em: https://www.python.org/
[17] https://github.com/akopytov/sysbench

DESCRIÇÃO	VERSÃO / MODELO
Desktop	Intel Core i7-3770, 16 GB RAM,
Sistema Operacional Linux	Ubuntu 16.04
Oracle VirtualBox	5.1.14
Docker	17.03.1-ce
Mysql Server	5.7
mysql/mysql-cluster	7.5
Python	2.7
Sysbench	0.4.12

Tabela 1 – Recursos/Ferramentas Utilizados

3.1 Descrição dos Cenários de Testes

Na Tabela 2, são descritas as características dos componentes que consistiu o cenário de testes.

Nome	Função	IP	CPU	Memória	SO
Gerente 1	Gerenciar o cluster	192.168.56.50	1	1024	Ubuntu 14.04
Gerente 2	Gerente Reserva	192.168.56.51	1	1024	Ubuntu 14.04
Gerente 3	Gerente Reserva	192.168.56.52	1	1024	Ubuntu 14.04
Normal 1	execução de serviço	192.168.56.53	1	1024	Ubuntu 14.04
Normal 2	execução de serviço	192.168.56.54	1	1024	Ubuntu 14.04
Normal 3	execução de serviço	192.168.56.55	1	1024	Ubuntu 14.04
Normal 4	execução de serviço	192.168.56.56	1	1024	Ubuntu 14.04

Tabela 2 – Descrição do Cenário de Teste

Em virtude da utilização de todas as ferramentas descritas anteriormente, criou-se um cenário para os testes. O funcionamento desse cenário é explicado em seguida:

No cenário de testes, definido na Tabela 2 e esquematizado na Fig. 7, observa-se a existência de três nodos gerentes e quatro nodos normais. Dos três nodos gerentes, apenas um será atribuído para o gerenciamento do *cluster*, e os outros dois nodos permanecerão de reserva para esse nodo gerente, funcionando como nodos normais. Quando o nodo gerente cair, um novo nodo ocupará sua função, mantendo assim a disponibilidade do serviço.

O funcionamento do fluxo de trabalho do cenário é descrito da seguinte forma: quando o serviço do *cluster* for requisitado , a requisição chegará ao nodo gerenciador, que por sua vez irá balancear a carga. Isso implica na verificação de como os nós normais estão. Por exemplo, se os nodos 1 e 2 já estiverem realizando alguma tarefa e os nodos 3 e 4 estiverem ociosos, o nodo gerenciador, irá encaminhar as requisições para os nodos ociosos.

Os quatro nodos normais serão destinados às distribuições das réplicas dos serviços, pelo nodo gerente. Por conseguinte, a operação de adicionar novos nodos ao *cluster* se mostrou uma tarefa simples com o *Docker Swarm*, visto que o mesmo possibilita, com facilidade, a escalabilidade e disponibilidade de um serviço, conforme detalhado anteriormente.

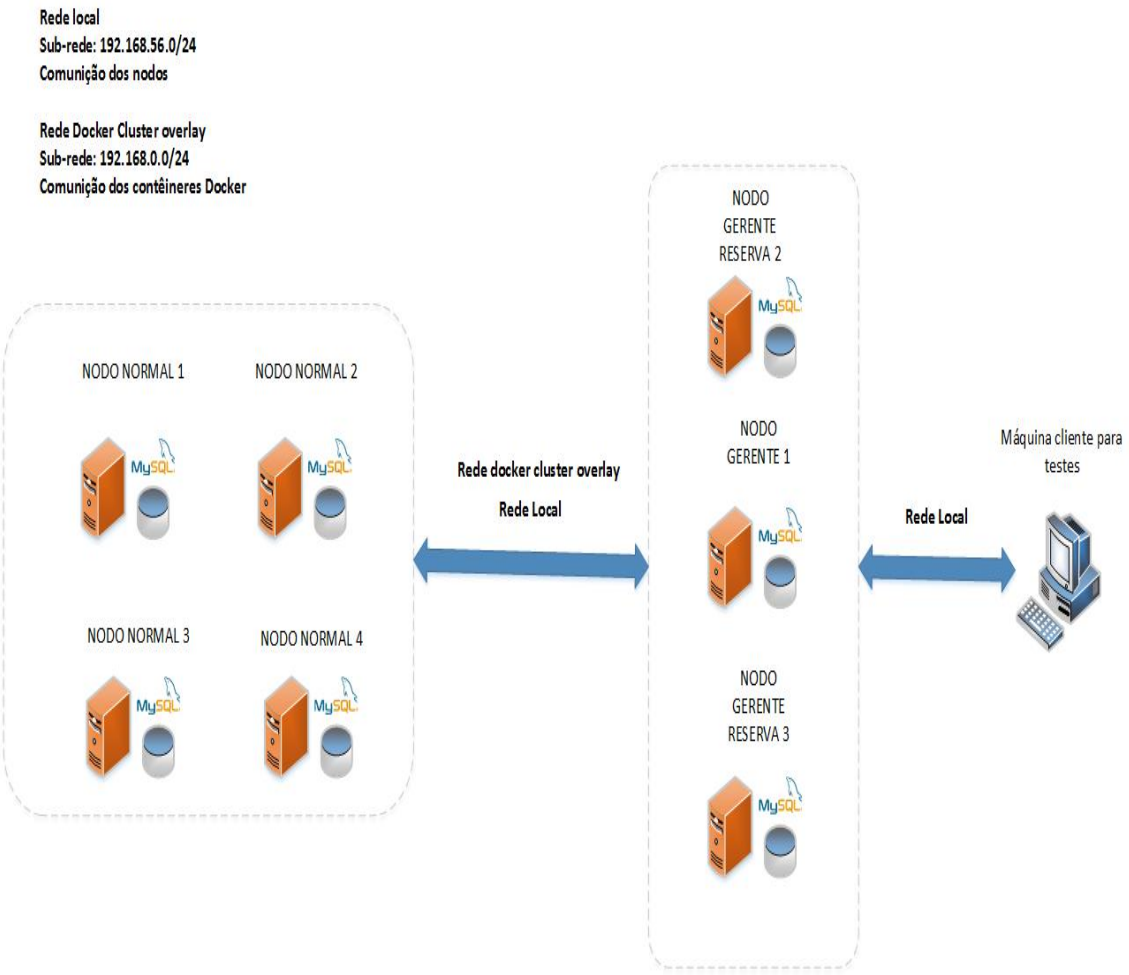

Figura 7 – Cenário de testes

O *cluster* é composto por máquinas com armazenamento próprio. Por causa disso, foi necessário implementar um mecanismo para manter a consistência dos dados. Primeiramente, a solução escolhida foi a implantação de um armazenamento distribuído. Assim sendo, depois de avaliar as ferramentas como: Infinit [18], Flocker [19] e GlusterFS [20]. A ferramenta Glusterfs foi escolhida, por sua simplicidade e facilidade de uso. Depois da implantação desta solução, configurada nos nodos do *cluster*, observou-se que os contêineres não conseguiam compartilhar nem as suas pastas de configuração nem os dados do MySQL. Isto porque, após a criação, eles não conseguiam iniciar, inviabilizando a execução no *cluster*.

A segunda solução utilizou-se de uma rede para manter a consistência dos dados, onde os armazenamentos dos nodos não eram compartilhados entre si, e sim replicados através da rede. Para essa solução, foi utilizada a imagem *Docker* do MySQL *cluster*, na qual há três tipos aplicações: 1)gerente, responsável por armazenar e administrar o *cluster*; 2)nodos de dados, aplicações para armazenar e distribuir os dados dos bancos; 3) nodo MySQL Server, aplicação

[18] Disponível em: https://infinit.sh/
[19] Disponível em: https://clusterhq.com/flocker/introduction/
[20] Disponível em: https://www.gluster.org/

para fazer as operações de SGBD. A partir da utilização do *engine* [21] ndbcluster, e a configuração do nodo gerente, o *cluster* conseguiu fornecer o serviço e manter a consistência dos dados.

Para os testes, implementou-se três cenários da arquitetura lógica do MySQL *cluster*: um com 2 nodos do MySQL Server; outro com 4 nodos MySQL Server; e por último uma composição de 7 nodos MySQL. A configuração de nodo gerente e dos nodos de dados não mudaram nos cenários, no qual há sempre um gerente e dois nodos de dados. A seguir esses cenários lógicos são mostrados:

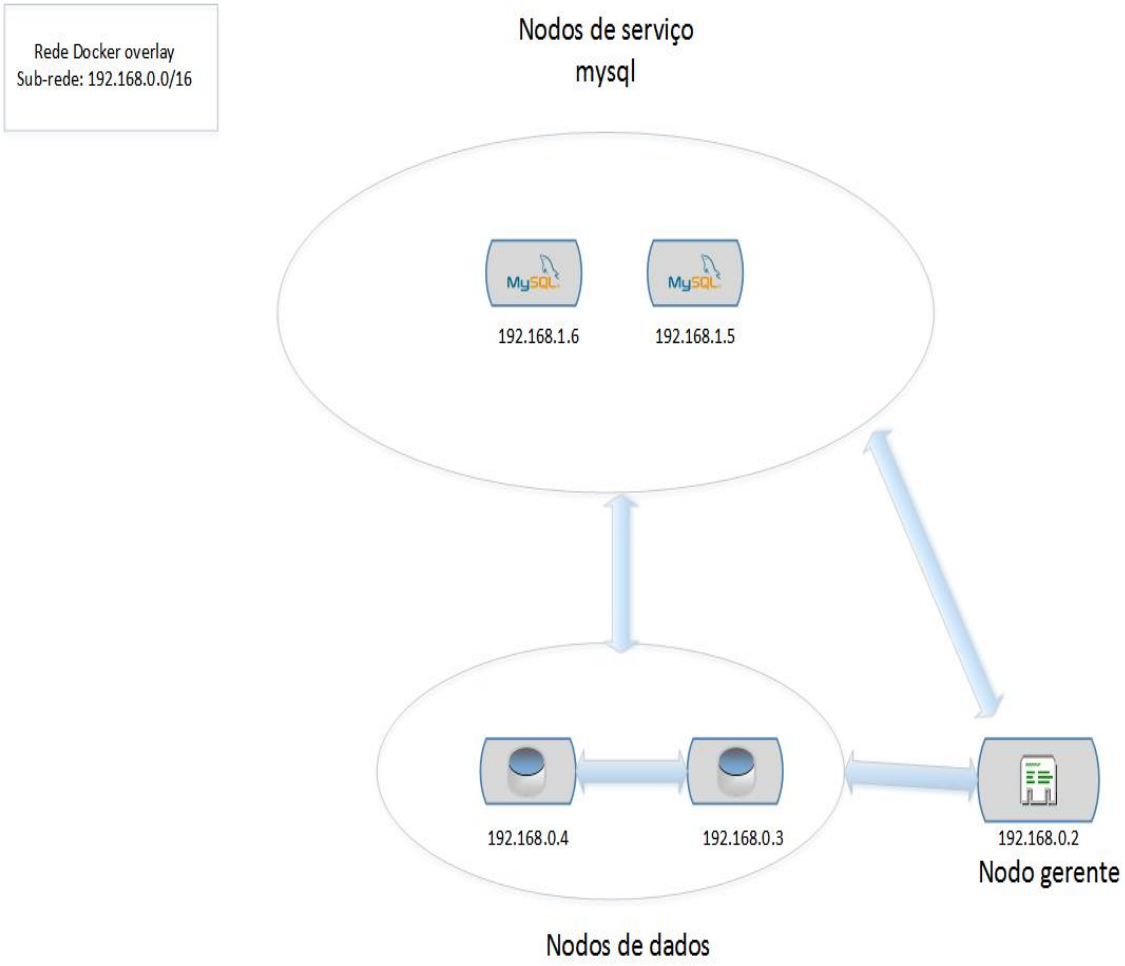

Figura 8 – Arquitetura lógica dos testes: 2 nodos MySQL

Na Figura 8 é configurado um contêiner como gerente do *cluster* MySQL, no nodo gerente 1 da arquitetura demostrada na Figura 7. Implementou-se os nodos de dados nos nodos gerentes 2 e 3 da mesma arquitetura. Após isso criou-se um serviço do MySQL Server, nos nodos normais da arquitetura de testes, com apenas duas instâncias.

Na Figura 9, a estrutura dos nodos de dados e do nodo gerente são as mesmas comentadas anteriormente, porém mudando a quantidade de réplicas que o serviço precisa executar, neste caso, as instâncias do serviço foram aumentadas para 4.

[21] Mecanismos de armazenamento no MySQL

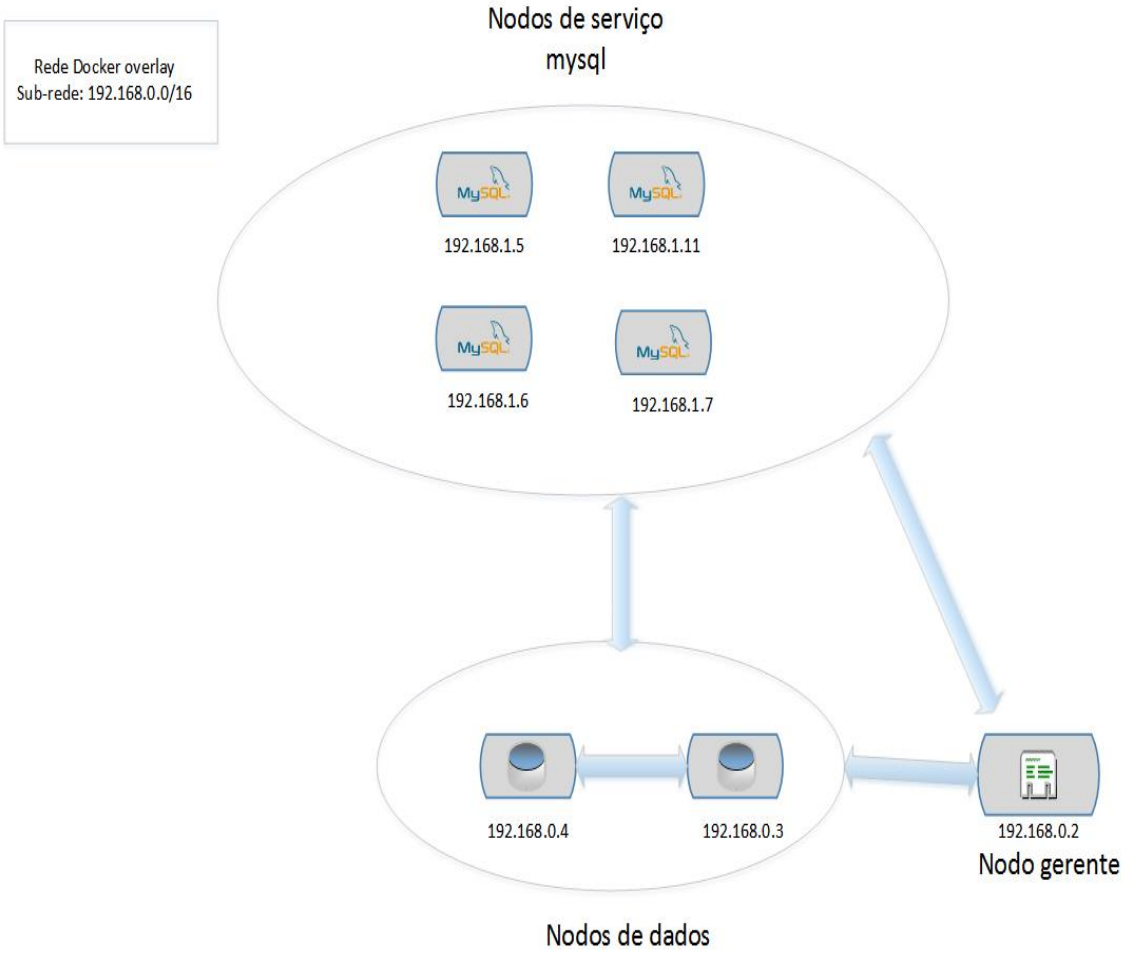

Figura 9 – Arquitetura lógica dos testes: 4 nodos MySQL

Na Figura 10, mantém-se a configuração dos nodos de dados e gerente. Nesse cenário utilizou-se de 7 réplicas do serviço, e também modificou-se a restrição das instâncias do serviço para executar apenas nos nodos normais.

De maneira, a permitir a comparação entre o desempenho do *cluster* e o modelo tradicional, foi implantado uma máquina com um processamento computacional igual a soma dos recursos do *cluster*, esses dados são demonstrados na Tabela 3 , em conjunto com a especificação da máquina que realizou os testes nos cenários. Para fazer os testes e capturar os resultados, implementou-se um *script* para gerar uma grande quantidade de requisições ao SGBD. Assim, é possível observar como foi o desempenho nos cenários(*cluster* e VM) sobre *stress*.

Nome	Função	IP	CPU	Memória	SO
Cliente	Fazer Testes	192.168.56.57	1	1024	Ubuntu 14.04
VM SGBD	execução de serviço	192.168.56.58	7	7168	Ubuntu 14.04

Tabela 3 – Descrição do cliente e VM para os testes

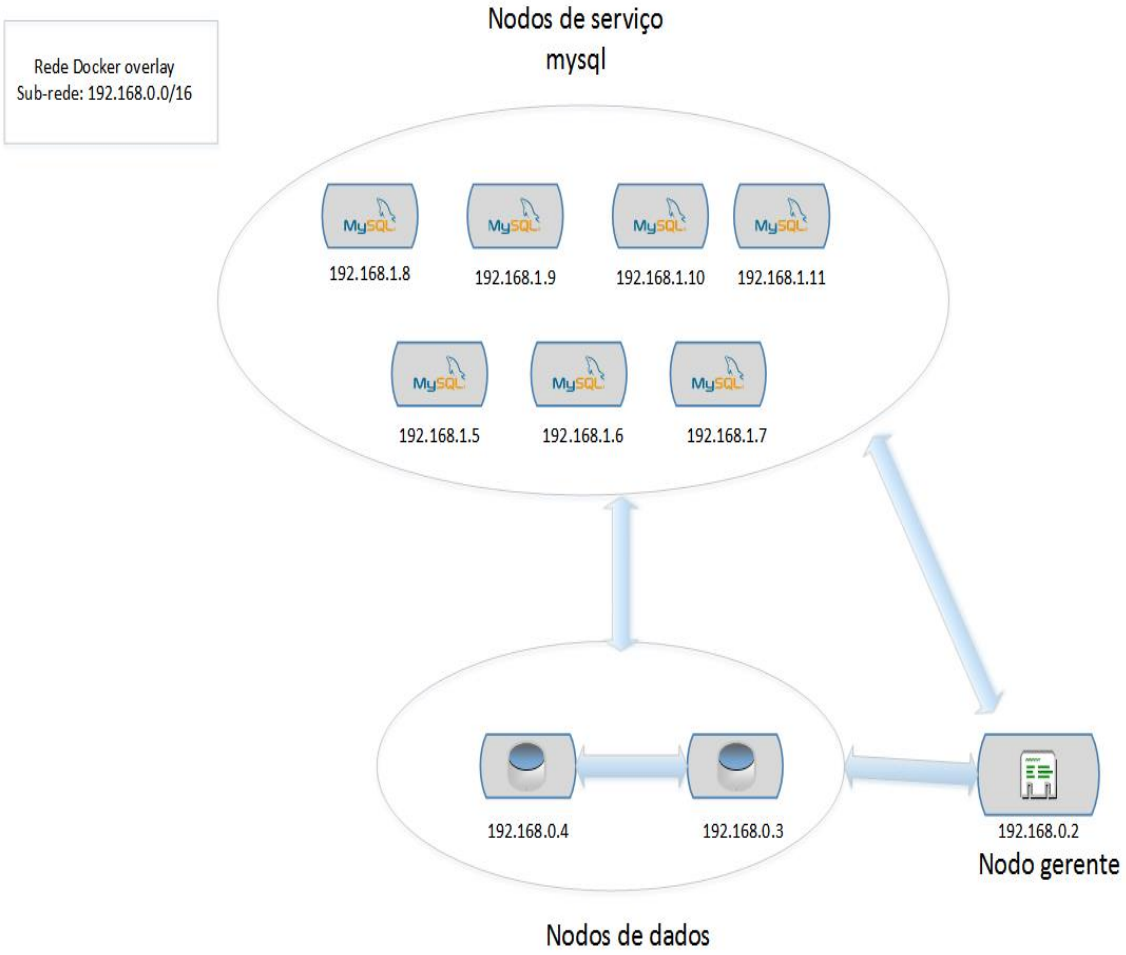

Figura 10 – Arquitetura lógica dos testes: 7 nodos MySQL

3.1.1 Sysbench

Por não conseguir gerar requisições suficientes para causar *stress* nos SGBDs, utilizando apenas o *script* em python, optou-se pela ferramenta de *benchmark* de SGBD, em conjunto com o *script*. As ferramentas avaliadas foram HammerDB [22], Apache Jmeter [23] e Sysbench. Por dificuldades de integração com *script*, as ferramentas HammerDB e Apache Jmeter não se mostraram uma boa opção, porém com o Sysbench a integração com *script* foi feita de forma fácil. Sendo assim, o Sysbench foi a ferramenta escolhida para gerar *stress* nos SGBDs. Além disso, outras funcionalidades ajudaram na escolha, sendo descritas no decorrer desta seção.

A ferramenta Sysbench permite a execução de testes de desempenho em um banco de dados e/ou em um sistema operacional. A partir dos teste, é possível verificar se o banco de dados e/ou o sistema operacional funcionará bem com alta carga de trabalho (KONDEL; GANPATI, 2015).

Por se tratar de uma ferramenta modular e *multithread*, o Sysbench permite ainda avaliar

[22] Disponível em http://www.hammerdb.com/
[23] Disponível em: http://jmeter.apache.org/

os parâmetros importantes de um sistema operacional e/ou de um banco de dados de acordo com a demanda. Desta forma, é possível verificar, de modo fácil e rápido, o desempenho do sistema operacional ou do banco de dados (POKHARANA; HADA, 2015).

De maneira geral, o Sysbench permite testar: desempenho de entrada e saída de arquivo, o desempenho de agendador de *threads*, a alocação de memória e velocidade de transferência, o desempenho de implementação de *threads* e o desempenho de servidor de banco de dados (POKHARANA; HADA, 2015). O funcionamento do Sysbench ocorre a partir de um banco de dados criado. Com isso, são criadas tabelas e são inseridos dados de acordo com os parâmetros[24] passados. Após a criação da base de dados, inicia-se uma quantidade de *threads* para fazerem requisições. De forma paralela, o Sysbench realiza as requisições, e o *script* python captura e armazena as métricas.

3.1.2 Métricas

Para analisar o banco de dados, utilizou-se as métricas mostradas na Tabela 4. A partir de uma quantidade de requisições, de tuplas e de *threads*, verificou-se como esses parâmetros se relacionavam com as variáveis de tempo de execução do teste, de porcentagem da memória utilizada e de porcentagem da cpu utilizada.

Métrica	**Descrição**
Quantidade de Requisições	valor utilizado pelo Sysbench para definir quantidade requisições utilizada no testes de desempenho nos SGBDs
Quantidade de tuplas do banco de dados	tamanho do banco de dados para os testes
Quantidade de *thread*	representa a quantidade de usuários solicitando o serviço dos SGBDs
Tempo de execução dos testes	quantidade de tempo que cada SGBD utilizou para tratar as requisições
Memória utilizada	quantidade de memória utilizada no momento dos testes
Porcentagem da CPU utilizada	quantidade de CPU utilizada durante os testes

Tabela 4 – Métricas Utilizadas nos Testes

3.1.3 Parâmetros do *Script*

Considerando a arquitetura implantada, e minunciosamente detalhada, o *script* implementado, recebe os seguintes parâmetros de entrada, conforme a Tabela 5. Esses valores foram escolhidos para tentar representar um momento no qual o SGBD tenha uma grande demanda para tratar. Além da demanda, o tamanho do banco de dados, foi escolhido com intuito de exemplificar uma base de uma universidade. Na quantidade de usuários, tentou-se intermediar

[24] os parâmetros estão relacionado com as métricas que são apresentadas no próximo tópico

uma quantidade razoável de usuários com os recursos, mostrados na Tabela 3, da máquina de testes.

Conjunto	Requisições	Tuplas do Banco de Dados	Quantidade de usuários
1	100000	100000	10

Tabela 5 – Parâmetros Utilizados nos testes

3.1.4 Pseudo-código dos Algoritmos Implementados

Para a realização dos testes , implementou-se, conforme o comentado anteriormente, um *script* na linguagem python, versão 2.7. O *script* utiliza os módulos de *csv* para arquivar os resultados , *threanding* para acionar funções simultaneamente, *commands* e *OS* para utilizar-se de chamadas do SO . Além do *script* para teste, implementou-se também um *script* para coletar as métricas de cpu e memória.

Algoritmo 1: Método principal

Entrada : Classe auxiliar Test
Saída : Vetor com o tempo de execução TVet

```
1  início
2      usu ← 10                          /* Define a quantidade de usuários */
3      tuplas ← 100000                   /* Define o tamanho do banco de dados */
4      req ← 100000                      /* Define a quantidade de requições */
5      IP ← 192.168.56.56                /* Informa o endereço para as requições*/
6      para i de 1 até 30 faça
7          Tvet[i] ← Test.testarBanco(req,tuplas,linhas,IP) /* Realiza os
8                                                                  testes */
9      fim
10     retorna Tvet                  /* Retorna os valores de tempo de execução
11                                          de cada teste */
12 fim
```

No Algoritmo 1, demonstra-se o método principal do *script* de execução do cliente, nas linhas 2, 3 e 4, foi configurados os valores dos testes, na linha 6 foi configurado um laço de repetição para executar os testes 30 vezes, dessa forma foi possível verificar os resultados do mesmo conjunto de teste, Tabela 5, e comparar seu comportamento no decorrer das repetições. Para testar os valores no SGBD, utilizou-se do método da classe Test: *testarBanco*, dessa forma foi criado um banco com os valores passados como parâmetros, e também definiu como e onde os testes foram feitos. O retorno do método, é um vetor com todos valores de tempo de execução das transações, de cada teste.

No Algoritmo 2, alguns parâmetros devem ser fornecido: quantidade de requisições que foram feitas ao banco de dados; de que tamanho foi criada a base de dados e por quantas *threads* foram executadas as requisições.

Algoritmo 2: Método de realização dos testes

Entrada : Quantidade de requições *REQ*, tamanho das tabelas *TUP*, quantidades de usuarios *US*, endereço de rede para os teste *IP*
Saída : Tempo de execução do teste *TE*

```
1  início
2      prepararBanco(TUP, IP)              /* Configura o tamanho do banco */
3      TE ← executarTeste(US,REQ,TUP,IP)
4      limparBanco(US, REQ, TUP, IP)       /*Limpa os dados inseridos no banco*/
5      retorna TE                          /* Retorna o tempo de execução dos
6                                              testes*/
7  fim
```

A saída do Algoritmo 2 foi o tempo de execução do teste realizado. A partir disso, foi possível demostrar o comportamento dos cenários ao tratar as requisições , e também fazer uma comparação de performance entre os cenários testados.

Algoritmo 3: Método para a coleta de dados: CPU

Entrada : Classe auxiliar Test
Saída : cpu utilizada

```
1  início
2      cpu ← 0.0                           /*Inicializa a variável*/
3      para i de 1 até 500 faça
4          saida ← Test.coletarCPU()       /* Verifica a cpu utilizada no
5                                              momento do teste*/
6          se saida > cpu então
7              cpu ← saida
8          fim
9          esperar(1)                      /* Espera 1 segundo para coletar novos
10                                             dados */
11     fim
12     retorna cpu                 /* Retorna o maior valor coletado da cpu */
13 fim
```

No Algoritmo 3 é demonstrado o método utilizado para coletar dados dos testes, relacionados a uso da cpu. Na linha 2 a variável cpu é inicializada para que não ocorra problemas na primeira utilização da mesma. Na linha 3 tem-se um laço de repetição para o tempo de execução do método execute enquanto os testes são feitos.

O método *coletarCPU* da classe Test, os valores de cpu são coletados, e a cada repetição é verificado se existe um valor maior de cpu, linha 6,no final do laço há um comando para esperar um segundo, linha 9 , isso permite que o monitoramento acompanhe o tempo de execução dos testes. Quando a execução da iteração terminar, o valor da variável cpu é retornado.

No Algoritmo 4, é apresentado o método utilizado para coletar a memória utilizada no momento testes. A estrutura do método é similar ao do Algoritmo 3, mudando apenas os valores

coletados, neste caso a memória utilizada durante os testes.

Algoritmo 4: Método para a coleta de dados: memória

```
   Entrada : Classe auxiliar Test
   Saída   : memória utilizada
 1 início
 2     memoria = 0.0                          / Inicializa a variável */
 3     para i de 1 até 500 faça
 4         saida ← Test.coletarMem() /* Verifica a cpu utilizada no
 5                                              momento do teste */
 6         if saída > memoria then
 7             memoria = saida
 8         end
 9         esperar(1)                  /* Espera 1 segundo para coletar novos
10                                                 dados*/
11     fim
12     retorna memoria                  /* Retorna o maior valor coletado da
13                                              memória */
14 fim
```

4 Resultados

Neste tópico, serão mostrados os resultados obtidos a partir do conjunto de teste definido no capítulo 3. Os dados encontram-se organizados em tabelas e gráficos, seguidos de uma breve análise estatística. Para tratar os dados, utilizou-se o *software* R [25] e a IDE RStudio [26], e para gerar os gráficos a ferramenta Gnuplot [27]. A seguir, os resultados apresentam-se através da arquitetura lógica do *cluster* MySQL, conforme o detalhamento minucioso no cap. 3.

4.1 Arquitetura Lógica com 7 Nodos MySQL Server

Nesta seção, serão apresentados os resultados obtidos com a configuração de 7 nodos MySQL. Cabe salientar que, nesta primeira configuração não limitou-se em quais nodos da arquitetura de testes, os contêineres MySQL Server poderiam executar. A seguir, esses resultados serão apresentados.

4.1.1 Tempo das Transações

A ferramenta Sysbench, utilizada para gerar carga no SGBD, retorna informações das transações feitas no SGBD. Um de seus parâmetros de retorno é duração de tempo para a realização de um teste. Neste caso os valores definidos na Tabela 5 do capítulo 3. Na Figura 11, os tempos de retornos tanto do SGBD instalado na VM quanto do SGBD do *cluster*, são demonstrados em seguida.

Observa-se na Figura 11, que os valores da VM estão sempre próximos de 12 minutos. os resultados mais altos obtidos estão nas execuções iniciais de 1 até a 5. Embora apresente valores altos nas execuções intermediárias de 10 até a 17, a VM apresentou, em determinados momentos, execuções de 6 até a 8, de 19 até a 21, e de 29 a 30, que chegaram a 9 minutos, sendo, esses resultados, valores bastante interessantes. Isso ocorreu porque, comparado ao *cluster*, o SGBD instalado na VM não precisou replicar seus dados, sendo possível também o cliente solicitar diretamente o serviço ao SGBD. O que não aconteceu no *cluster*, pois no mesmo as requisições foram redirecionadas do gerente para os nodos normais.

Na Figura 11, são analisados os resultados obtidos pelo *cluster*. As execuções iniciais de 1 até a 3, tiveram valores muito altos, seguidos de uma diminuição de valores. Aproximadamente, entre 22 e 18 minutos. Esses valores altos, obtidos pelo *cluster*, são explicados em virtude da solução de consistência dos dados do mesmo, visto que requerem que os valores armazenados nos nodos da arquitetura lógica sejam replicados para os nodos de dados e para os outros nodos da mesma arquitetura. Esses resultados estão associados a este cenário de teste, caso as execuções fossem feitas em ambiente de produção, em um *data center* com alta disponibilidade, os valores

[25] Disponível em:https://www.r-project.org/
[26] Disponível em:https://www.rstudio.com/
[27] Disponível em:http://www.gnuplot.info/

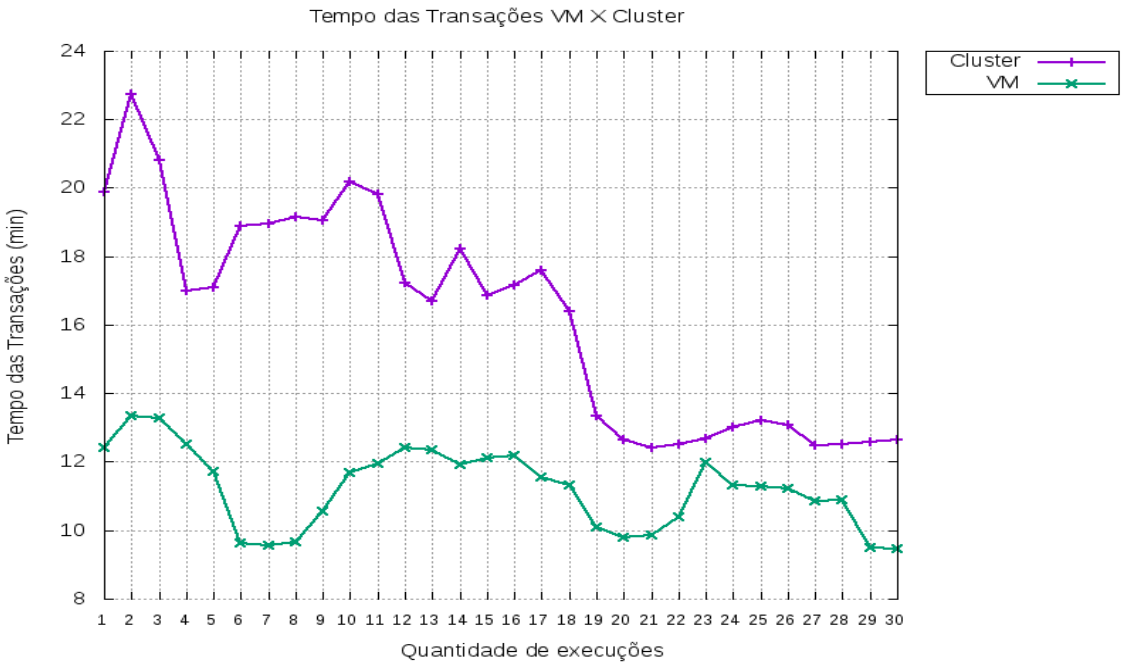

Figura 11 – Tempo das Transações com 7 Contêineres MySQL

obtidos seriam bem melhores. Da execução 9 até a 17, são demonstrados valores perto dos resultados mais altos, de 20 minutos, porém nas 10 últimas execuções, de 20 até a 30, há uma grande diminuição do tempo de execução das transações, com valores próximos a 12 minutos.

Conclui-se que essa configuração do *cluster*, comparado com os valores obtidos do SGBD instalado na VM , não apresenta resultados satisfatórios, pois os mesmos ficaram muito acima dos valores obtidos da VM. Porém essa configuração garante alta disponibilidade do serviço, já que existem 7 nodos do serviço MySQL da arquitetura lógica executando. Dessa forma, caso algum dos nodos pare de funcionar, o *Docker Swarm* irá criar um novo contêiner do serviço para substituição ou simplesmente as requisições serão direcionadas para os outros nodos.

Na Tabela 6, os dados estatísticos de ambos cenários são mostrados. Verifica-se que os valores da VM tiveram uma média de 11.23 minutos, enquanto que o *cluster* teve 16.23 minutos. A partir da média e a quantidade de execuções do conjunto de teste, foram necessárias 5 horas para realização das 30 execuções com a VM, e 8 horas com o *cluster*. O desvio padrão da VM mostra que seus valores se distanciaram um pouco da média, porém para o *cluster* essa variação foi um pouco maior.

	Tempo de Execução VM	Tempo de Execução Cluster
Média	11.23	16.23
Mediana	11.32	16.91
Desvio Padrão	1.16	3.17
Intervalor de confiança	10.80 - 11.66	15.05 - 17.41
Maior Valor	13.34	22.73
Menor Valor	9.47	12.40

Tabela 6 – Resultados dos testes estatísticos do tempo das transações

4.1.2 Utilização de CPU

Nesta seção, será apresentada a quantidade de cpu utilizada no momento dos testes. Os valores apresentam-se em porcentagem. Os gráficos estão separados em uso de cpu dos gerentes do *cluster* dos nodos normais. A seguir os resultados serão demonstrados.

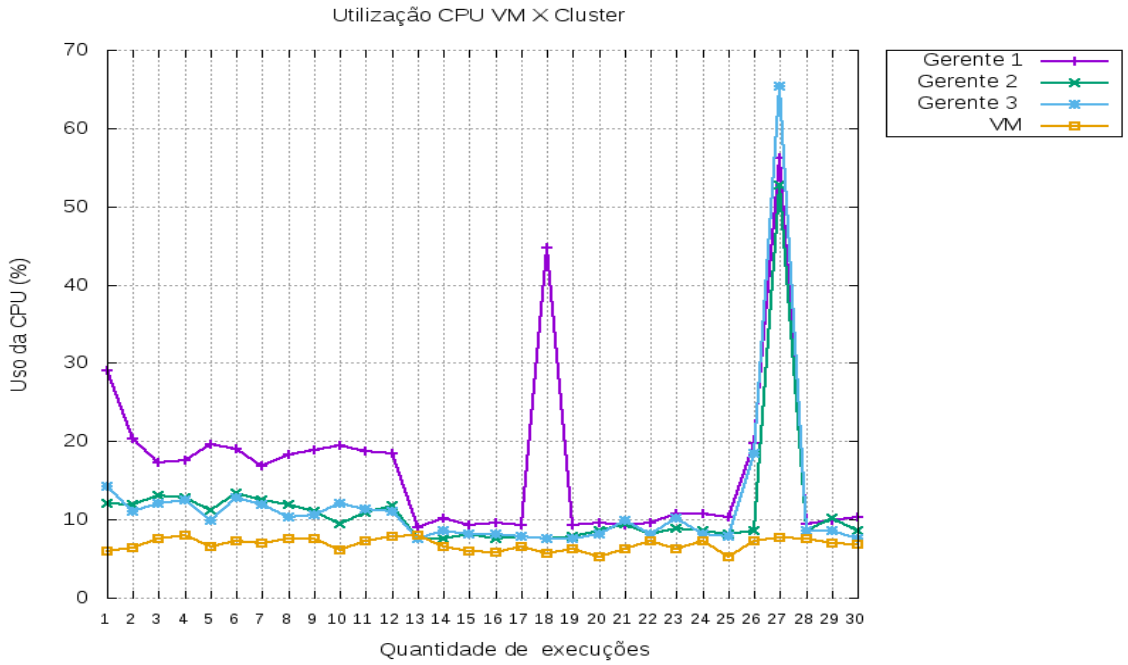

Figura 12 – Utilização de CPU VM X Gerentes do Cluster com 7 Contêineres MySQL

Na Figura 12, observa-se que os valores da VM não se modificaram no decorrer das execuções, mantendo-se sempre próximos de 10%. Os resultados dos gerentes do *cluster*, na maioria dos testes ficam um pouco acima de 10%, no entanto, em alguns momentos, execuções 18 e 27, os valores chegaram a 45 ou acima de 50 %. Isso pode ter sido gerado por algum problema interno de gerenciamento do *cluster*. Observa-se, na Figura 13, que os valores da cpu utilizada pelos nodos normais acompanharam os resultados dos gerentes, com exceção das execuções iniciais. Isso aconteceu por causa do balanceamento do nodo gerente. Isto porque as execuções inicias de 1 até 10 não foram, bem distribuídas, até chegar na execução 12.

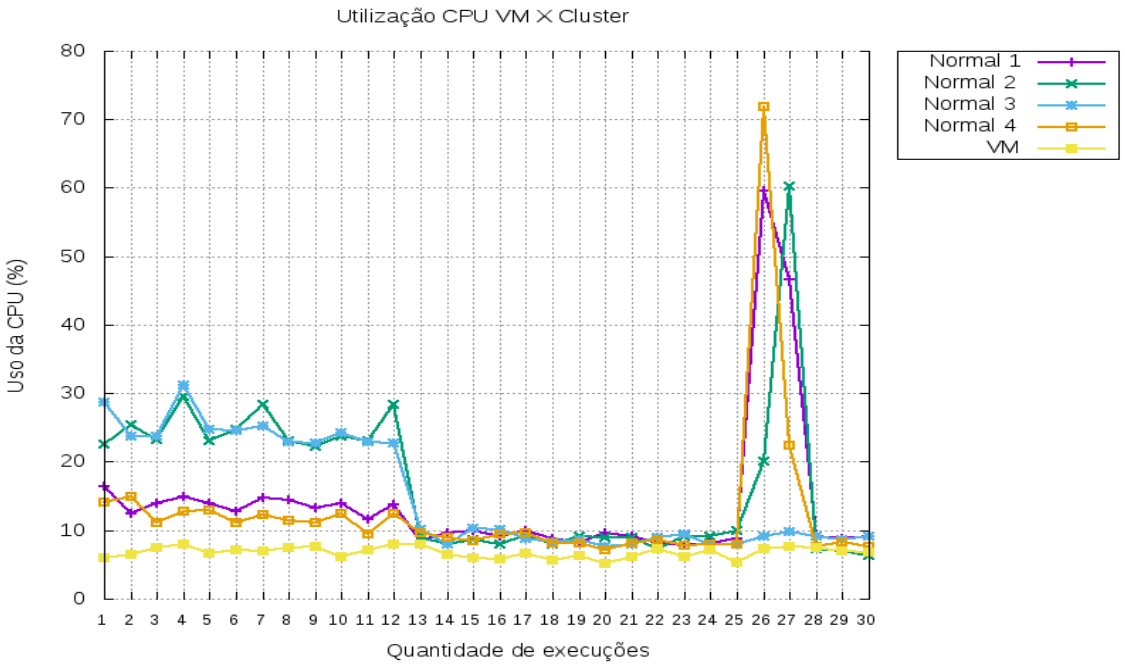

Figura 13 – Utilização de CPU VM X Nodos Normais do Cluster com 7 Contêineres MySQL

Conclui-se que o processamento da VM sempre se mantém estável e abaixo dos valores do *cluster*. Esses resultados são explicados pelas atividades extras que o *cluster* tem que fazer para manter a consistência dos dados, pois os nodos estão sempre se comunicando para a replicação dos dados.

Na Tabela 7, os resultados da VM demonstram maior estabilidade e pouca variação dos valores. O *cluster* apresenta valores com intervalos grandes, porém esse comportamento ocorre por causa dos valores utilizados para gerar a estatística, já que esses dados foram obtidos de todos os nodos do *cluster*. Desse modo, os valores de um nodo podem estar bem abaixo de outro nodo, dependendo do momento.

	Uso CPU VM %	Uso CPU Cluster %
Média	6.77	14.06
Mediana	6.85	9.90
Desvio Padrão	0.80	10.40
Intervalor de confiança	6.48 - 7.07	12.64 - 15.47
Maior Valor	8.00	71.90
Menor Valor	5.20	6.30

Tabela 7 – Resultados dos testes estatísticos do uso da CPU

4.1.3 Utilização de Memória

Nesta seção, serão demonstrados os resultados da utilização da memória nos cenários(VM e *cluster*). O modelo de apresentação e discussão será semelhante ao tópico anterior.

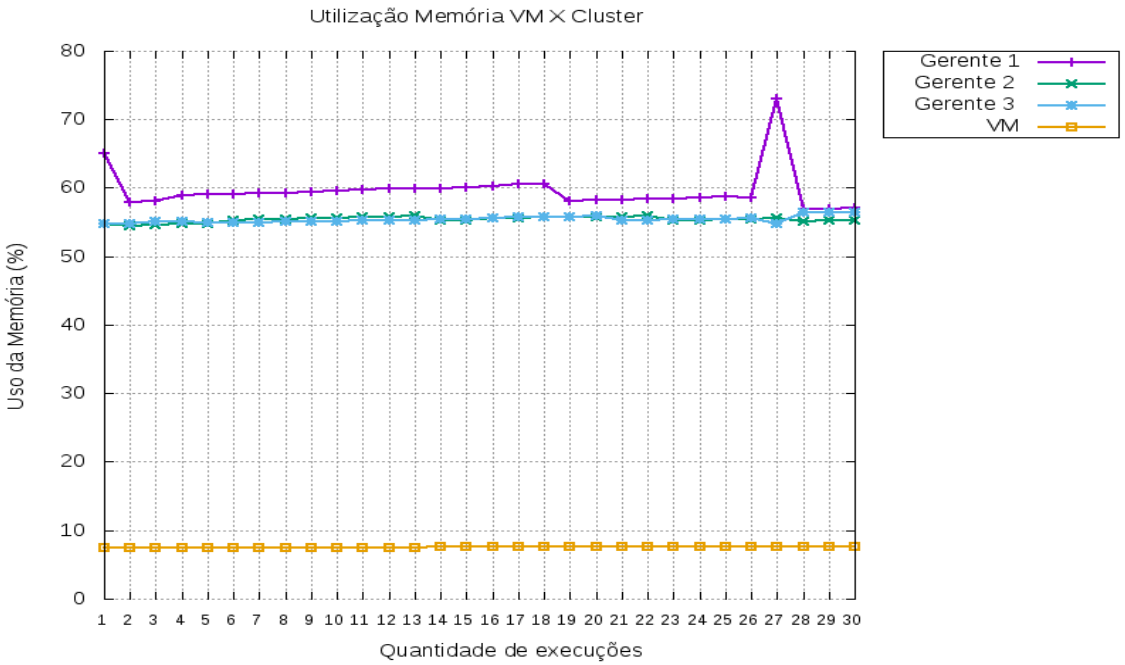

Figura 14 – Utilização de Memória VM X Gerentes do Cluster com 7 Contêineres MySQL

Nas Figuras 14 e 15, os dados dos gerentes do *cluster* se mantêm sempre acima de 55%, e, em alguns momentos, na execução 27, são alcançados valores próximos de 75%. Os nodos normais tiveram resultados bem acima do uso de memória dos gerentes, com os valores mais baixos perto de 75%, e com valores mais altos chegando a 95%. Esses resultados ocorreram por causa da troca constante de dados, durante as execuções, entre os nodos da arquitetura lógica. Desse modo, os nodos permaneceram sempre armazenando dados, tanto para atualizar suas base de dados quanto para distribuir dados na rede. Os dados coletados da VM, mostraram-se estáveis, sempre próximos de 10%. Esses valores baixo são explicados por sua base de dados ser somente local, assim a memória é utilizada apenas para tratar as requisições.

Percebe-se que no *cluster* a memória é, portanto, muito utilizada, principalmente nos nodos normais. Fazendo um comparativo com a VM, obtêm-se valores muito elevados com a soma do recursos do *cluster*. Desse jeito, essa configuração da arquitetura lógica do *cluster* utiliza bem mais da memória do que a VM. Conforme comentado anteriormente, isso acontece por causa da replicação dos dados que *cluster* realiza para manter a consistência de dados entre seus nodos.

Na Tabela 8, são apresentados os valores estatísticos dos dois cenários. Nos quais se obtiveram valores mais altos e dispersos para o *cluster*. Conforme comentado anteriormente os valores da VM permaneceram bem a baixo do *cluster*, e sempre próximo da média de 7.56%.

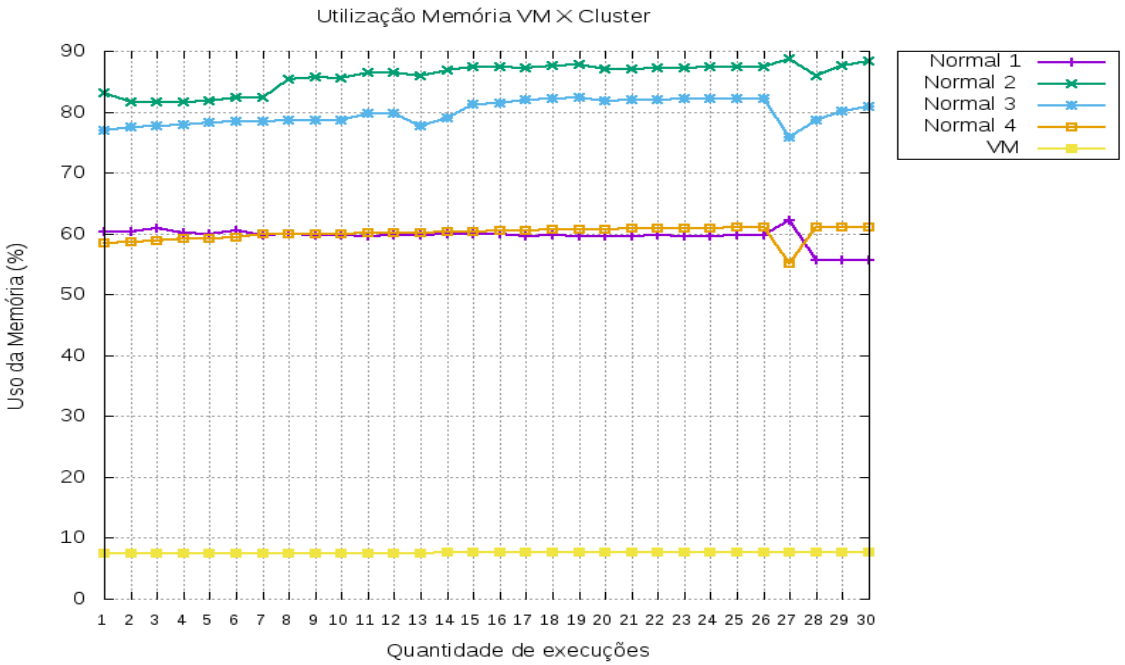

Figura 15 – Utilização de Memória VM X Nodos Normais do Cluster com 7 Contêineres MySQL

	Uso Memória VM	Uso Memória Cluster
Média	7.56	65.09
Mediana	7.6	59.77
Desvio Padrão	0.5	11.64
Intervalor de confiança	7.54 - 7.58	61.51 - 66.08
Maior Valor	7.61	88.64
Menor Valor	7.48	54.49

Tabela 8 – Resultados dos testes estatísticos do uso da memória

4.2 Arquitetura Lógica com 4 Nodos MySQL Server

Na arquitetura lógica com 4 nodos MySQL Server, executou-se o *script* de teste definido capítulo 3, que por sua vez executou o teste de stress 30 vezes. Configurou-se os parâmetros para os testes conforme a Tabela 5 do capítulo 3. Os resultados são demonstrados em seguida.

4.2.1 Tempo das Transações

A cada teste, o Sysbench retornou quanto tempo foi necessário para sua execução. A quantidade de requisições configuradas, em um cenário definido, neste caso a quantidade de usuários e tamanho do banco de dados influência diretamente os resultados.

Verifica-se na Figura 16, que nos testes feitos na máquina virtual, nos quais as primeiras execuções, compreendem da execução 1 até a 4, os valores foram os mais altos próximos a 13 minutos, seguidos de uma grande queda no tempo de execução. Observa-se também que os resultados sempre permaneceram perto de 11 minutos de execução, mesmo que, em alguns

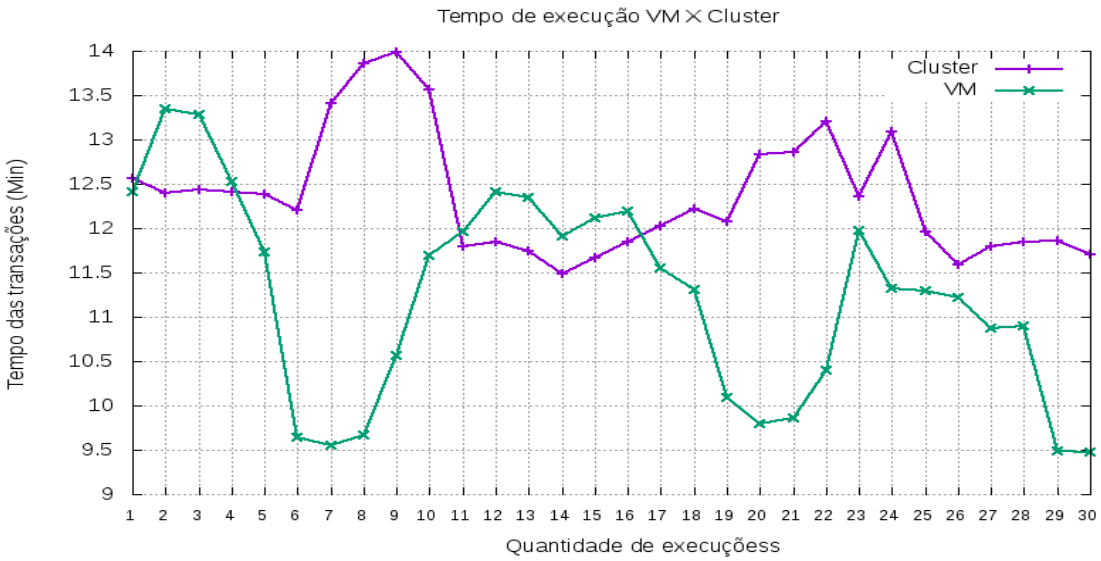

Figura 16 – Tempo das Transações com 4 Contêineres MySQL

momentos, o tempo de execução tenha sido bem inferior, a partir dos 9,5 minutos, nas execuções de 6 até a 8, de 19 até a 21, e de 29 e 30. As observações foram as mesma a respeito dos resultados da VM, comentados anteriormente na seção 4.1.1

Ainda na Figura 16 , percebe-se que nos testes no *cluster*, foi mantida uma média de 12 minutos nas primeiras execuções de 1 até a 6, seguida de um aumento do tempo de execução, nas execuções de 6 até a 10. Após o aumento, o tempo de execução ficou inferior às primeiras execuções, estabilizando-se, até outra crescente no tempo, nas execuções de 19 até a 24. Esses resultados se dão por causa do cenário de teste, caso as execuções fossem feitas em ambiente de produção, em um *data center* com com uma rede SAN[28], os resultados seriam bem melhores. Depois, nas últimas execuções, o tempo voltou para os resultados próximos aos valores mais baixo registrados anteriormente, de 11 minutos e meio. Nesse cenário, os resultados foram melhores do que o apresentado na seção 4.1.1, mostrando que o desempenho está ligado a quantidade de nodos no *cluster* da estrutura lógica.

Conclui-se que mesmo não mantendo um tempo de execução sempre próximo a um valor, a VM tem resultados melhores que o *cluster*. Enquanto que no *cluster*, os resultados dos testes iniciais são melhores do que da VM, em alguns momentos, nas execuções de 6 até a 10, os valores se mostram muito acima dos da VM. Os valores altos do *cluster* acontecem por causa da replicação de dados entre os nodos, e pelo gerente do nodo ter que intermediar as requisições, ao invés das requisições serem direcionadas diretamente para o serviço. Nessa configuração lógica do *cluster* os resultados se mostram mais estáveis e com tempo inferiores a dos resultados

[28] rede de armazenamento

mostrados anteriormente na seção 4.1. *cluster*

	Tempo de Execução VM	Tempo de Execução Cluster
Média	11.23	12.37
Mediana	11.32	12.21
Desvio Padrão	1.16	0.69
Intervalor de confiança	10.80 - 11.66	12.11 - 12.63
Maior Valor	13.34	13.99
Menor Valor	9.47	11.49

Tabela 9 – Resultados dos testes estatísticos do tempo de transação

Na Tabela 9, os resultados estatísticos das execuções são apresentados. Os valores da VM em comparação com o *cluster*, foram menores, mostrando que seus resultados foram melhores. Porém no *cluster* os valores foram mais próximos da média, como mostra o desvio padrão. Essa diferença entre os resultados mostra que a VM, mesmo tendo resultados melhores, oscila muito, enquanto que no *cluster* seus valores seguem próximos da média, mesmo que mais alto que os da VM.

4.2.2 Utilização de CPU

No momento de execução dos testes, capturou-se a utilização da cpu em cada teste, tanto na VM como nos nodos do *cluster*. Para facilitar a visualização, criou-se dois gráficos, nos quais um comparou os resultados da VM e os gerentes do *cluster*, e o outro da VM e os nodos normais do *cluster*. A seguir os resultados são apresentados.

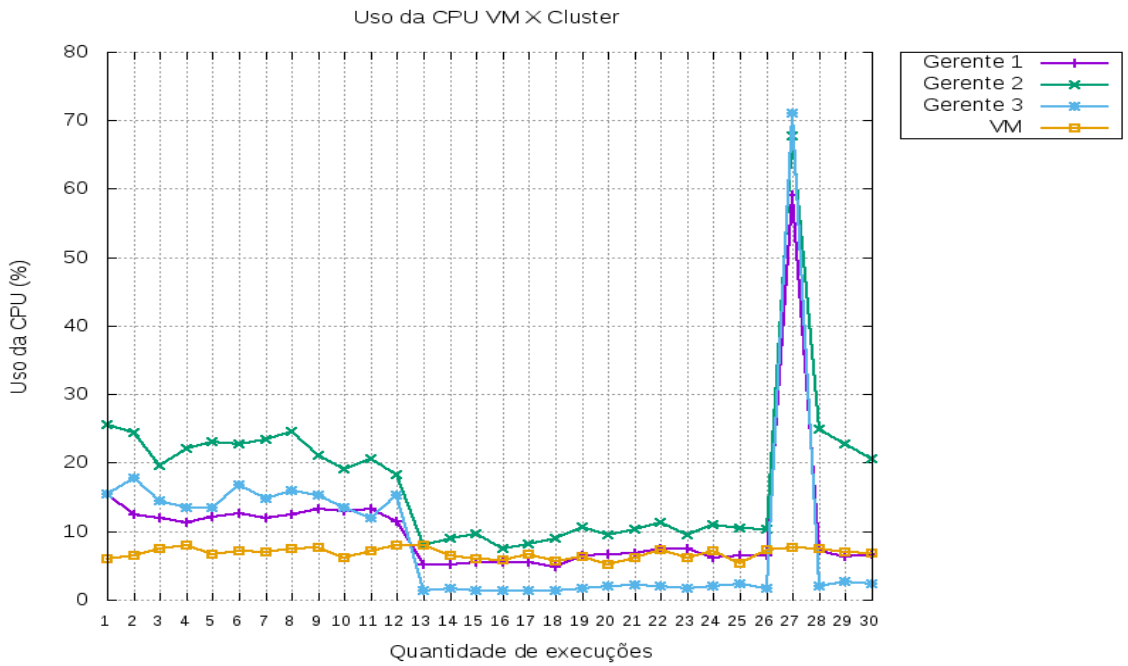

Figura 17 – CPU utilizada VM X Gerentes do *cluster* com 4 Contêineres MySQL

Obeserva-se na Figura 17, que os valores da VM foram sempre perto 10%, mantendo o uso da cpu em um nível baixo durante os testes. O conjunto de teste, fez 100.000 requisições ao SGBD, mostrando que essa máquina virtual não teve dificuldades em tratar dessa demanda. Tendo em consideração também as observações feitas na seção 4.1.2, pois os dados são os mesmos.

Novamente na Figura 17, percebe-se que os gerentes utilizaram uma quantidade semelhante de cpu nos testes iniciais, e que em seguida o uso da cpu começou a diminuir, esses resultados estão relacionados ao balanceamento da demanda entre os nodos do *cluster*. Nos testes finais os gerentes tiveram um pico de uso de cpu, que pode ter sido ocasionado por algum problema na gerência interna do *cluster*.

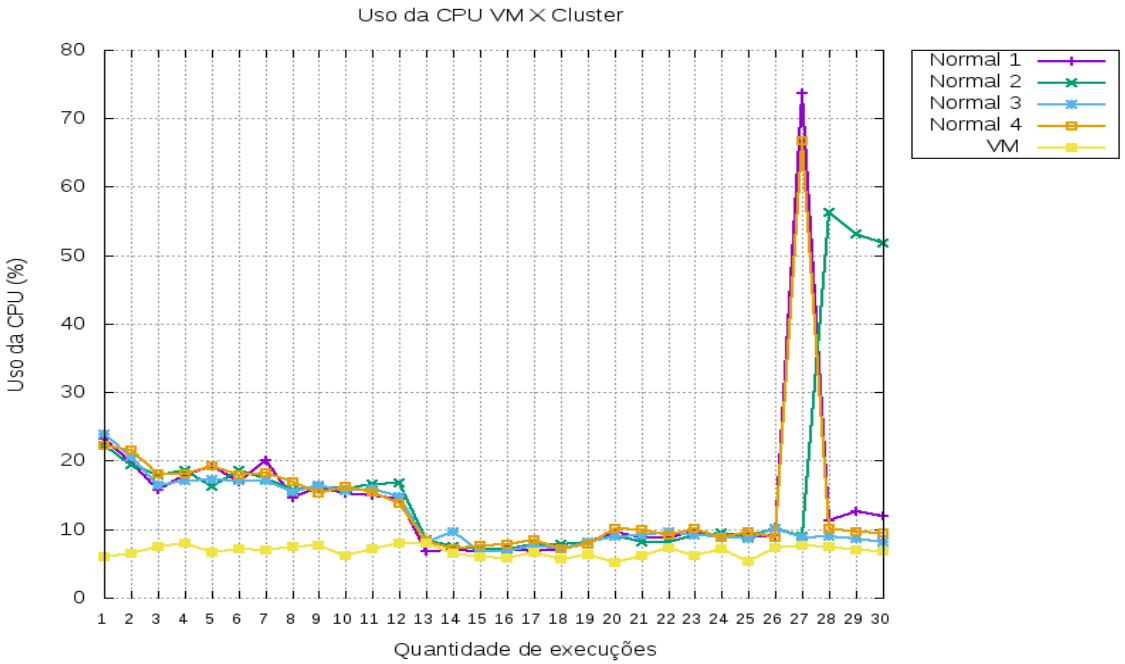

Figura 18 – CPU utilizada VM X Nodo normais do *cluster* com 4 Contêineres MySQL

Na Figura 18, utiliza-se os mesmos dados da VM, mostrados na Figura 17 . Na parte do *cluster* percebe-se que nos testes iniciais, nas execuções de 1 até a 11, o uso da cpu foi maior, seguido da diminuição da utilização da cpu. Tal como nos nodos gerentes mostrados anteriormente na Figura 17, com a diferença de que o uso da memória dos nodos normais foram menores que dos nodos gerentes. Nos teste finais, nas execuções de 26 até a 28, ocorreu um aumento do uso da cpu, mostrando possível problema no funcionamento do *cluster*, o que não foi tratado no teste, sendo considerado um *outlier*.

Conclui-se que os valores obtidos pela VM são melhores do que os do *cluster*, pois os resultados sempre ficam próximos aos 10%, desde os testes iniciais até os últimos testes, enquanto que no *cluster* os resultados dos testes iniciais começam altos, seguidos de uma diminuição. Vale notar que, nas últimas execuções, de 26 até a 28, o *cluster* apresentou um

aumento do uso da cpu, negativamente afetando a comparação para o *cluster*. Alguns nodos do *cluster* conseguem resultados menores ou igual a VM, porém a maioria dos nodos tem resultados maiores, esses resultados acontecem porquê o *cluster* precisa realizar atividades extras para manter a consistência dos dados, e isso consequentemente gera maior uso da CPU.

	Uso CPU VM	Uso CPU Cluster
Média	6.77	13.58
Mediana	6.85	10.20
Desvio Padrão	0.80	11.39
Intervalor de confiança	6.48 - 7.07	12.04 - 15.13
Maior Valor	8.00	73.70
Menor Valor	5.20	1.40

Tabela 10 – Resultados dos testes estatísticos do uso da CPU

Na Tabela 10, os resultados estatísticos do uso da CPU são mostrados, tanto da VM quanto do *cluster*. A média de utilização de cpu da VM foi menor do que a do *cluster*, desse modo a VM conseguiu alcançar resultados melhores que o *cluster*. Enquanto que a VM conseguiu manter valores sem grandes mudanças no decorrer dos testes, o *cluster* apresentou várias mudanças de aumento e diminuição do uso da cpu. Isso ocorreu porque os dados utilizados foram de todos os nodos do *cluster*. Desse modo, os valores de um nodo variam de um para o outro, gerando esses valores discrepantes.

4.2.3 Utilização de memória

A partir dos dados coletados no momento da execução dos testes, criou-se gráficos e tabelas para demonstrar esses resultados, seguidos da comparação de cenários e de comentários sobre. Os resultados dos dados de memória utilizada são mostrados em seguida.

Verifica-se na Figura 19, que o uso da memória por parte da VM não teve grande modificação no decorrer das execuções. O valor coletado demonstrou resultados próximos de 10%, mantendo-se sempre a baixo desse valor. As observações são as mesma da seção 4.1.3.

Ainda na Figura 19, os valores coletados dos gerentes do *cluster* demonstraram uma variação para cada gerente. assim o gerente 1 foi que utilizou menos memória, enquanto que os resultados dos gerentes 2 e 3 mostraram valores de 10 a 15% maiores. Esses valores, podem ser explicados pelos contêineres rodados nos nodos gerentes 2 e 3. Os serviços desses contêineres executaram nodos de dados do *cluster* MySQL, por isso, foi preciso utilizar mais memória para armazenar e replicar os dados para outros nodos.

Na Figura 20, obeserva-se que os valores do *cluster* foram mais altos que o da VM. Isso ocorre porque, além de fazer as operações de SGBD, o *cluster* também replica seus dados entre os nodos da arquitetura lógica.

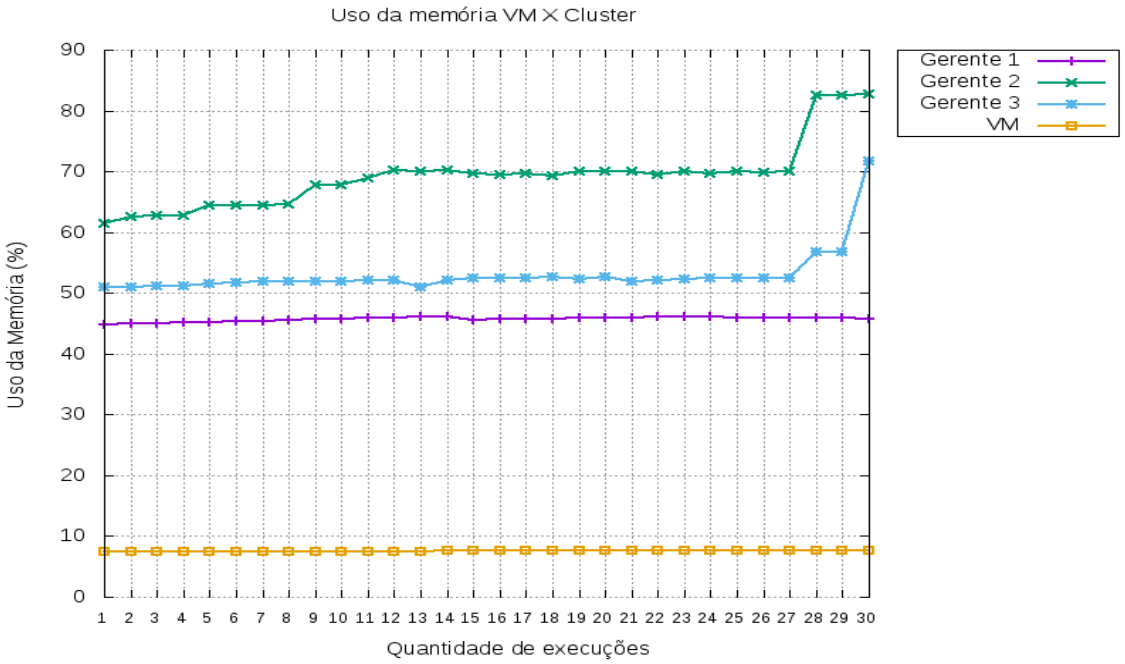

Figura 19 – Memória utilizada VM X Nodos Gerentes do *cluster* com 4 Contêineres MySQL

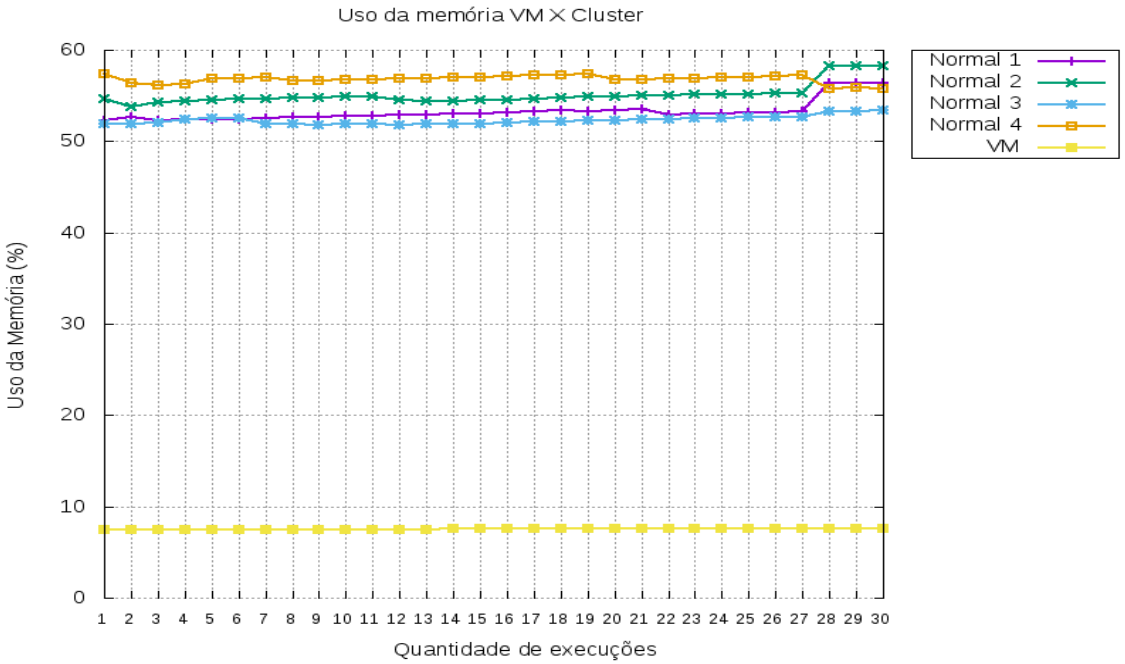

Figura 20 – Memória utilizada VM X Nodos normais do *cluster* com 4 Contêineres MySQL

Na Tabela 11, os resultados estatísticos do uso de memória são mostrados. Percebe-se que a média da VM foi bem inferior a média do *cluster*, e também os valores do *cluster* foram mais dispersos, como mostra o desvio padrão.

	Uso memória VM	Uso de memória *Cluster*
Média	7.56	55.05
Mediana	7.6	53.23
Desvio Padrão	0.5	7.09
Intervalo de confiança	7.54 - 7.58	54.09 - 56.01
Maior Valor	7.61	82.73
Menor Valor	7.48	44.76

Tabela 11 – Resultados dos testes estatísticos do uso da memória

4.3 Arquitetura Lógica com 2 Nodos MySQL Server

Nesta seção, serão apresentados os resultados obtidos com a arquitetura lógica com 2 nodos MySQL Server. Em seguida os resultados serão demonstrados.

4.3.1 Tempo das Transações

A partir dos resultados retornados do Sysbench, foi gerado gráficos e cálculos estatísticos do tempo das transações. Esses resultados são demonstrados a seguir.

Na Figura 21, são apresentados os resultados do tempo de execução das transações nos cenários do *cluster* e da VM. Observa-se que os valores do *cluster* mantiveram uma estabilidade no decorrer das execuções, e que seus resultados foram sempre próximos de 11 minutos. No cenário da VM, os teste iniciais apresentaram os valores mais altos, próximos de 13 minutos, nas execuções 1 até a 3. Porém, em algumas execuções, de 6 até a 8, de 19 até a 21, e de 29 até a 30, os valores caíram para 9 minutos e meio. As execuções da VM demonstraram resultados que oscilaram entre 9 e 12 minutos, no decorrer das execuções, como comentado anteriormente.

Conclui-se que, apesar dos valores ótimos, como 9 minutos e meio, A VM não mantém esses valores. No *cluster*, os resultados têm uma maior estabilidade, permanecendo em vários momentos abaixo dos valores da VM. Os resultados dos dois cenários se intercalam entre as execuções. As vezes, os resultados melhores são da VM, outras do *cluster*. Além disso, o *cluster* consegue manter uma consistência dos resultados, uma vez que seus valores não mudam muito no decorrer das execuções.

Na Tabela 12, os dados estatísticos são apresentados. Os dados dos cenários foram bem próximos, mudando mais o desvio padrão. Os resultados do *cluster* não se alteraram muito, enquanto que na VM os resultados oscilaram demasiadamente.

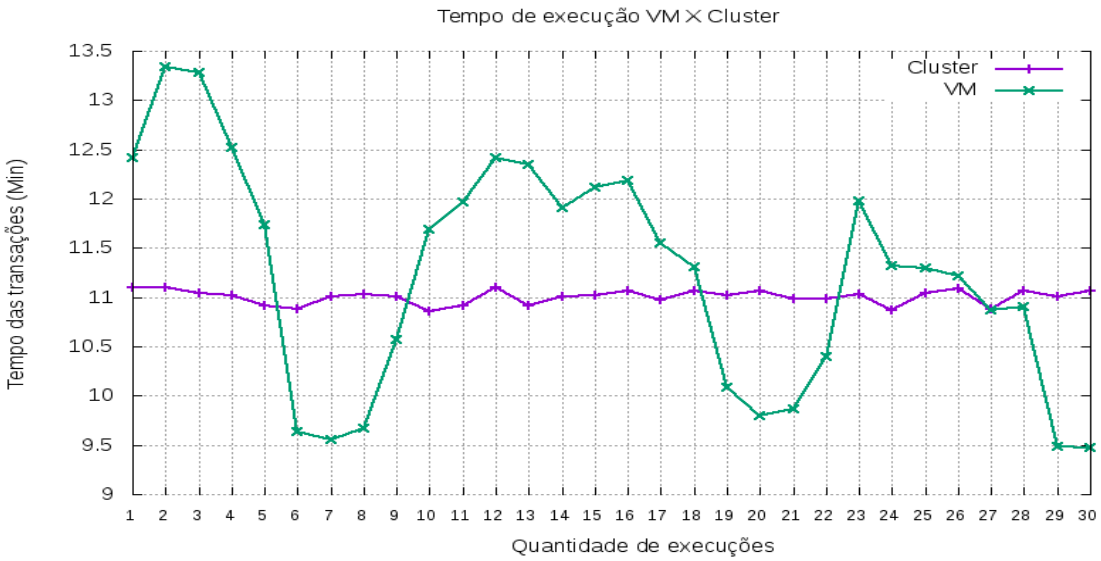

Figura 21 – Tempo das Transações VM X *cluster* com 2 Contêineres MySQL

	Tempo de Execução VM	Tempo de Execução *Cluster*
Média	11.23	11
Mediana	11.32	11.02
Desvio Padrão	1.16	0. 07
Intervalo de confiança	10.80 - 11.66	10.98 - 11.03
Maior Valor	13.34	11.10
Menor Valor	9.47	10.86

Tabela 12 – Resultados dos testes estatísticos do tempo de transação

4.3.2 Utilização de CPU

A seguir serão apresentados os resultados alcançados de cpu no momento da execução dos testes.

Na Figura 22, os resultados dos nodos gerentes foram superiores ao da VM. Na qual os valores dos gerentes do *cluster* 2 e 3 oscilaram entre 16 e 18% de uso da cpu, no decorrer das execuções, enquanto que no gerente 1 os valores ficam sempre próximos de 12%, esses resultados acontecem pelo *cluster* ter que replicar seus dados entre seus nodos . Na Figura 23, em dois nodos, normal 1 e 3, ocorreram algo parecido com o comentado anteriormente na Figura 22, pois os resultados oscilaram de 18 a 25%, no decorrer das execuções. Esses valores são explicados pelas tarefas extras que o *cluster* faz para manter a consistência dos dados. Na VM os valores ficam sempre próximos de 10%.

Na Tabela 13, os valores estatísticos do *cluster* apresentaram valores maiores que o da VM. Isso ocorre em virtude das execuções com valores superiores em todas as execuções, e

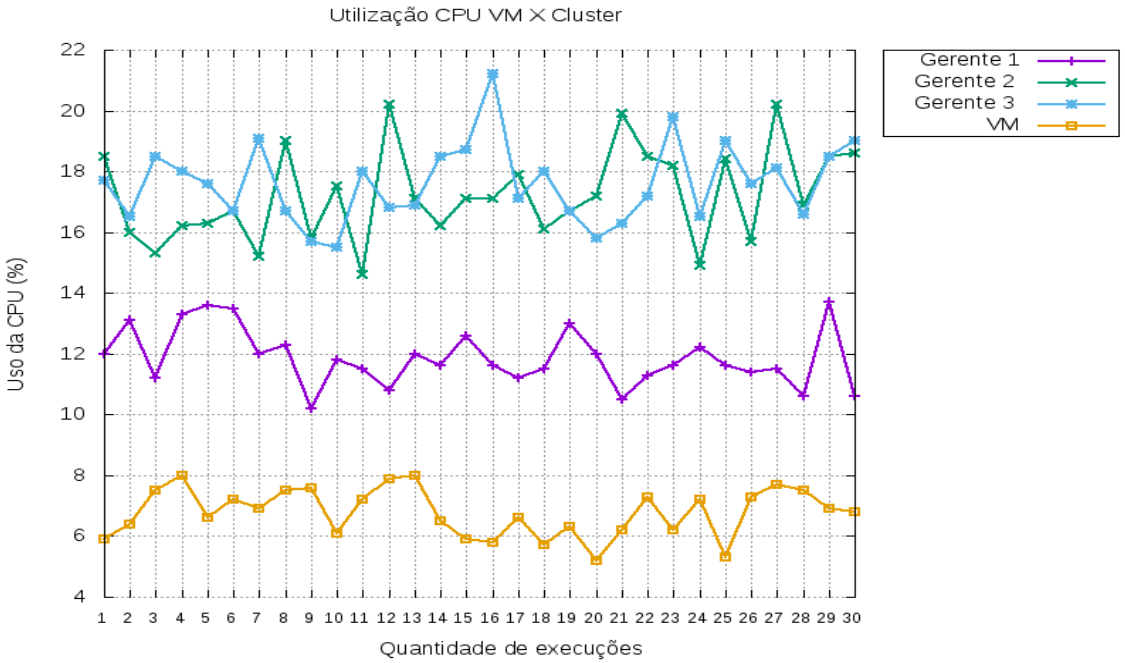

Figura 22 – Utilização CPU VM X Gerentes do *cluster* com 2 Contêineres MySQL

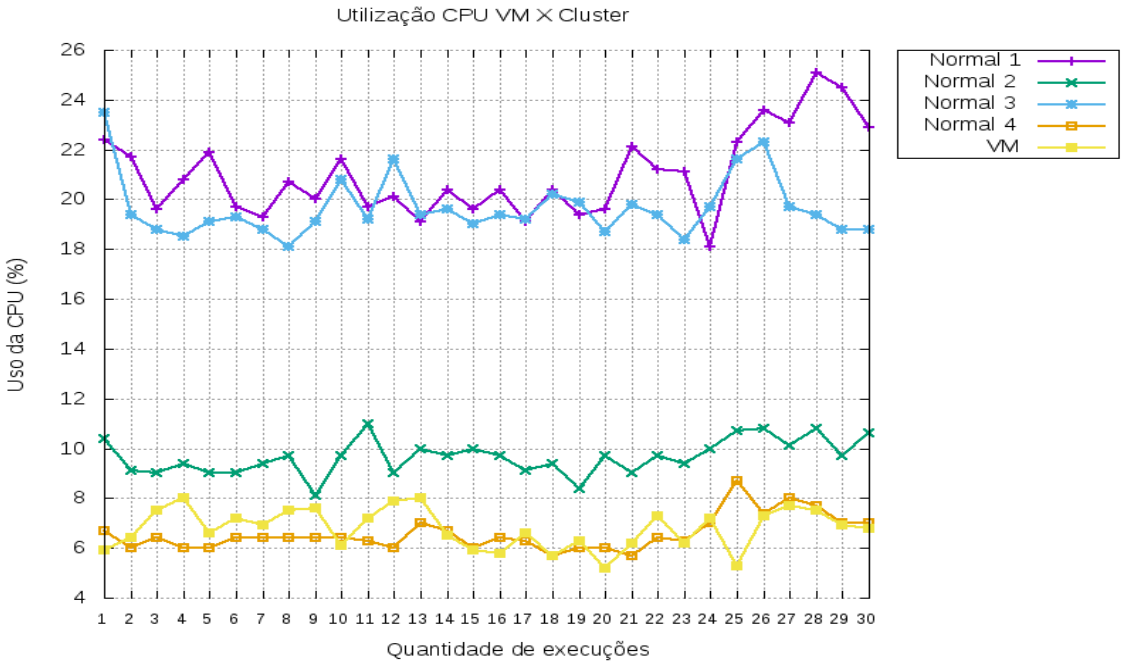

Figura 23 – Utilização CPU VM X Nodos Normais do *cluster* com 2 Contêineres MySQL

também pela utilização dos dados de todos os nodos do *cluster*.

	Uso CPU VM %	Uso CPU Cluster %
Média	6.77	14.79
Mediana	6.85	16.40
Desvio Padrão	0.80	5.20
Intervalor de confiança	6.48 - 7.07	14.08 - 15.50
Maior Valor	8.00	25.10
Menor Valor	5.20	5.70

Tabela 13 – Resultados dos testes estatísticos do uso da CPU

4.3.3 Utilização de Memória

Nesta seção, serão apresentados os resultados, da utilização da memória do *cluster* e da VM, obtidos dos testes.

Na Figura 24, são demonstrados os valores dos gerentes do *cluster*. Nele, os gerentes 2 e 3 utilizaram cerca de 65 a 70%. Isso acontece devido os nodos de dados da arquitetura lógica estarem nesses nodos, precisando, assim, de mais memória para gerenciar os dados no *cluster*. O gerente 1 manteve os valores próximos de 50%. Na VM, os resultados se mantiveram sempre abaixo de 10%. As observações são as mesmas da seção 4.1.3.

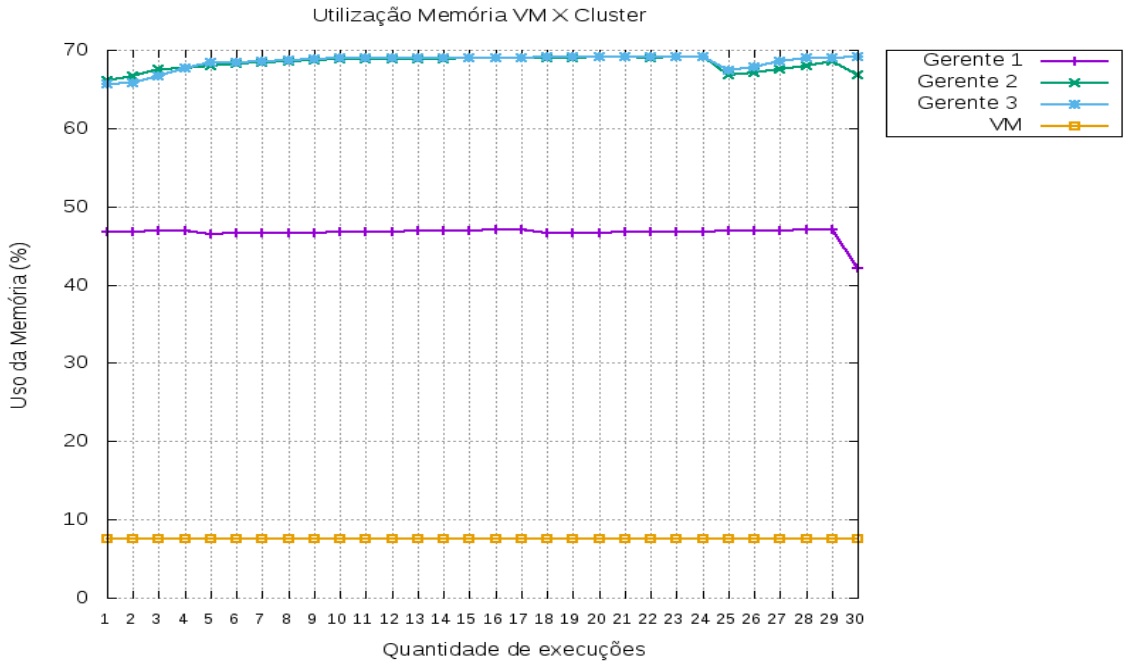

Figura 24 – Utilização Memória VM X Gerentes do *cluster* com 2 Contêineres MySQL

Observa-se na Figura 25, que os resultados da nodos normais também se apresentaram com valores altos, estando próximos de 55 e 60%. Esses valores são explicados pelos processos que os nodos precisam fazer para replicar os dados no *cluster*.

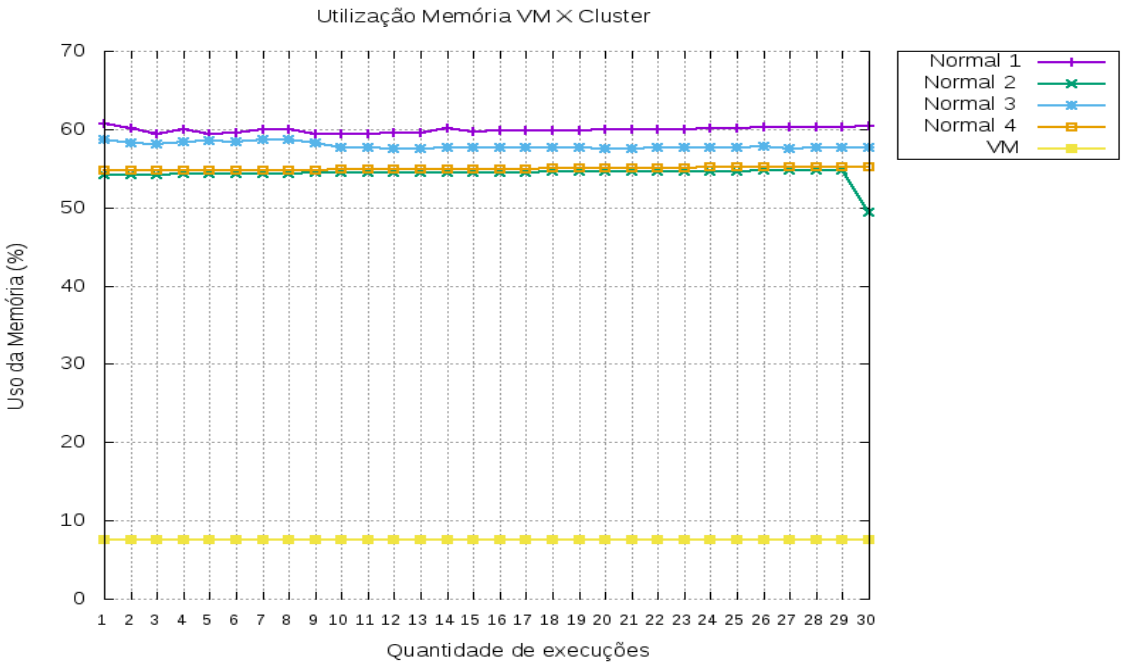

Figura 25 – Utilização Memória VM X Nodos Normais do *cluster* com 2 Contêineres MySQL

Na Tabela 14 são apresentados os cálculos estatísticos. Os valores obtidos do *cluster* se mostraram bem acima dos resultados da VM, tanto na média quanto na dispersão dos valores para a média como mostra o desvio padrão de 7.34 do *cluster*.

	Uso memória VM	Uso de memória *Cluster*
Média	7.56	58.71
Mediana	7.6	57.67
Desvio Padrão	0.05	7.34
Intervalor de confiança	7.54 - 7.58	57.51 - 59.71
Maior Valor	7.61	69.13
Menor Valor	7.48	42.18

Tabela 14 – Resultados dos testes estatísticos do uso da memória

5 Considerações Finais

O intuito desse trabalho consistiu em implementar um *cluster* SGBD com microsserviços. Para a implantação no modelo de microsserviço, utilizou-se o recurso de contêineres. Desse modo, criou-se autonomia entre os serviços, já que os contêineres não dividiam nenhuma dependência ou algo que acopla-se os serviço fortemente. A partir da comunicação através da rede e a independência do serviços, a utilização de contêineres para provê serviços, se mostrou uma opção adequada para utilização do modelo de microsserviços.

A utilização de microsserviços implementados em contêineres, mostrou-se uma solução plausível para implantação de serviços. Em relação aos problemas (criação, configuração e gerência de serviços) citados no capítulo 1 do trabalho, depois de entender os conceitos visto no capítulo 2 e implementar os mesmos no capítulo 3, observou-se que esse modelo de implantar serviço, provê uma solução rápida e simples dos problemas.

Outros benefícios vistos, foi as opções de configurar serviços através de arquivos e separar uma aplicação em módulos, e cada módulo funcionando em um contêiner diferente, com isso é viável implantar serviços em diferentes ambientes e escalonar serviços de maneira otimizada.

A utilização de *cluster* com microsserviços , pode ser utilizado para algum momento que uma instituição ou empresa tiver aumento de demanda de algum serviço, como por exemplo, em uma instituição de ensino, na época vestibular ou final de período, lançamento e visualização das notas. Além do *cluster* balancear a carga nos nós, ainda é viável escalonar o serviço até a demanda passar, e depois voltar para estado anterior de quantidade de serviços. Além disso a implantação de novos serviços, sem afetar os serviços em funcionamento, pode ser feito de maneira ágil e simples, e ainda replicado para outros campus da universidade.

Para testar e avaliar as novas tecnologias e modelos de implantação de serviço propostos nesse trabalho, e assim verificar se são viáveis para ambientes de produção, implementou-se um ambiente de teste descrito no capítulo 3. Por esse cenário utilizar um *cluster*, faz-se necessário o gerenciamento de recursos e agendamento do tratamento das solicitações. A ferramenta escolhida para tal tarefa, foi *Docker Swarm*, pois com essa ferramenta é possível criar um serviço e distribui-lo pelo *cluster*, sem a necessidade de configuração do serviços em outros nodos que não sejam o gerente, além disso o escalonamento do serviço criado, não envolveu uma configuração complexa, precisando apenas informar pelo gerente, a partir de um comando, a nova configuração do serviço.

As dificuldades se concentraram na percepção de separação ou fragmentação do serviço, gerando mais configurações para o funcionamento do ambiente. Outro problema encontrado inicialmente consistia no *cluster* apenas está escalando o serviço, sem fazer o controle dos dados que estão sendo gerenciado, isso implica na inconsistência do dados de um nodo para outro. Para resolver esse problema utilizou-se de replicação do dados pela rede (também testou-se

uma solução com um volume distribuído, porém a mesma apresentou erros nos contêineres no momento da execução do serviço). Para essa solução utilizou-se do mecanismo de *cluster* do próprio SGBD MySQL, no qual utiliza-se de nodos de dados, para centralizar e distribuir os dados entre os nodos. Escolheu-se essa solução, pois ao executar o serviço no *cluster*, os contêineres apresentaram funcionamento correto e consistência dos dados.

Para demonstrar a viabilidade do serviço em *cluster*, configurou-se 3 cenários lógicos, no qual muda-se apenas a quantidade de nodos do MySQL Server. A escolha desse cenários são motivados pelos teste iniciais, no qual notou-se a diferença de desempenho ao modificar a quantidade de nodos do MySQL Server, os nodos citados são referente a arquitetura lógica e não a nível da arquitetura onde o *Docker Swarm* funciona. Dessa forma comparou-se os resultados do 3 cenários com um SGBD instalado em uma VM, com recursos iguais a soma dos recursos do *cluster*. Assim os cenários com 4 e 2 nodos MySQL Server, mostraram-se viáveis, já que os mesmos conseguiram resultados próximos ao da VM, e ainda proviam alta disponibilidade do serviço e facilidade de escalonamento.

A partir dos resultados, a implantação do *cluster* SGBD com microsserviços, se mostrou uma solução viável para ambiente de produção de uma instituição ou empresa. Esse modelo, provê alta disponibilidade e facilidade de escalonamento. Utilizando a ferramenta do *Docker*: *Docker Swarm*, não foi necessário utilizar diferentes ferramentas para implantação do *cluster*. Isso demonstra a simplicidade e facilidade dessa ferramenta para provimento de serviço.

5.1 Trabalhos futuros

Pretende-se como trabalhos futuros:

- Realizar um estudo comparativo entre as ferramentas de gerenciamento de *cluster* em contêineres: *Docker Swarm* e *Kubernetes*;
- Modificar a estrutura dos cenários utilizados no trabalho, e também definir novos parâmetros para os testes. Utilizando como base para os testes, um ambiente de produção;
- Integrar diferentes tecnologias para provimento do serviço SGBD.

Referências

BENZ, K.; BOHNERT, T. M. Impact of pacemaker failover configuration on mean time to recovery for small cloud clusters. *IEEE*, 2015. 26

DEITEL, H. M.; DEITEL, P. J.; CHOFFNES, D. R. *Sistemas Operacionais*. 3. ed. São Paulo: Pearson, 2005. 17

EMER, B. Implementação de alta disponibilidade em uma empresa prestadora de serviços para internet. 2016. 9, 24

GROßMANN, M.; KLUG, C. Monitoring container services at the network edge. *IEEE*, 2017. 27, 28, 29

JARAMILLO, D. V. N. D. Leveraging microservices architecture by using docker technology. *IEEE*, 2016. 17, 19, 20, 22, 23

KANG MICHAEL LE, S. T. H. Container and microservice driven design for cloud infrastructure devops. *IEEE*, 2016. 24

KHAN, M.; TOEROE, M.; KHENDEK, F. *Comparing Pacemaker with OpenSAF for Availability Management in the Cloud*. 2017. 25

KONDEL, A.; GANPATI, A. Evaluating system performance for handling scalability challenge in sdn. *IEEE*, 2015. 35

MOUAT, A. *Usando Docker*. 1. ed. São Paulo: Novatec Editora, 2015. 14, 27, 28

NAIK, N. Applying computational intelligence for enhancing the dependability of multi-cloud systems using docker swarm. *IEEE*, 2016. 24, 28

NEWMAN, S. *Building Microservices: Designig Fine-Grained Systems*. 1. ed. United States of America: O'REILLY, 2015. 17, 18, 19, 20

PEREIRA, R. B. d. O. Alta disponibilidade em sistemas gnu/linux utilizando as ferramentas drbd, heartbeat e mon. 2005. 14

PERKOV NIKOLA PAVKOVIC, J. P. L. High-availability using open source software. *IEEE*, 2007. 24, 26

POKHARANA, A.; HADA, R. Performance analysis of guest vm's on xen hypervisor. *IEEE*, 2015. 36

POSADAS, J. V. *Application of Mixed Distributed Software Architectures for Social-Productive Projects Management in Peru*. 2017. 9, 19

RIASETIAWAN, M.; ASHARI, A.; ENDRAYANTO, I. *Distributed Replicated Block Device (DRDB) Implementation on Cluster Storage Data Migration*. 2015. 24, 25

SALAH M. JAMAL ZEMERLY, C. Y. Y. M. A.-Q. Y. A.-H. T. Performance comparison between container-based and vm-based services. *IEEE*, 2017. 9, 22, 23

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Fundamentos de Sistemas Operacionais*. 8. ed. Rio de Janeiro: LTC, 2010. 16, 17, 21, 22

TANENBAUM, A. S. *Sistemas operacionais modernos*. 3. ed. São Paulo: Pearson Prentice Hall, 2009. 14, 20, 21, 22

TANENBAUM, A. S.; STEEN, M. V. *Sistemas Distribuídos: princípios e paradigmas*. 2. ed. São Paulo: Pearson, 2007. 9, 16

TSAI, P.-H. et al. *Distributed Analytics in Fog Computing Platforms Using TensorFlow and Kubernetes*. 2017. 28, 29

ZAMINHANI, D. Cluster de alta disponibilidade atravÉs de espelhamento de dados em mÁquinas remotas. 2008. 9, 26, 27

ZHANG, X. L. J.; K., D. Performance characterization of hypervisor- and container-based virtualization for hpc on sr-iov enabled infiniband clusters. *IEEE*, 2016. 22

6 Apêndice

6.1 Instalação e Configuração do Ambiente

O processo de instalação das ferramentas é detalhado nas seções subsequentes.

6.1.1 Instalando o Docker

Primeiramente atualize os repositório do sistema com comando a baixo:

1. *sudo apt-get update*

Como a versão utilizada nos nodos foi a ubuntu 14.04. Recomenda-se a instalação de pacotes extras.

1. *sudo apt-get install*
2. *linux-image-extra-$(uname -r)*
3. *linux-image-extra-virtual*

Modo de instalação escolhido foi por repositório. Para isso é necessário instalar pacotes que permitam a instalação via repositório:

1. *sudo apt-get install*
2. *apt-transport-https*
3. *ca-certificates*
4. *curl*
5. *software-properties-common*

Para configurar o repositório do docker, utilizou-se o comando abaixo:

1. *sudo add-apt-repository*
2. *"deb [arch=amd64] https://download.docker.com/linux/ubuntu*
3. *$(lsb_release -cs)*
4. *stable"*

A versão escolhida foi a docker CE, edição da comunidade. Antes de instalar o docker, atualize os índices dos pacotes.

1. *sudo apt-get update*

Em seguida instale o docker:

1. *sudo apt-get install docker-ce*

6.1.2 Configurando Cluster Docker Swarm

Comandos para criação do *cluster* swarm são mostrados em seguida:

1. *docker swarm init –listen-addr IP-NODO-MESTRE –advertise-addr IP-NODO-MESTRE*

Para adicionar um nodo é necessário configurar uma chave no nodo a ser adicionado.

1. *docker swarm join –token=564879321459562*

6.1.3 Configurando o Serviço Mysql no Cluster

Para configuração do Mysql no cluster, criou-se três contêineres e um serviço. Sendo um contêiner gerente e dois contêineres para gerenciar dados, e o serviço criado foi do Mysql Server. Abaixo é mostrado como configurar esses componentes:

1. *docker network create –attachable –subnet=192.168.0.0/24 cluster*

Primeiramente é necessário criar e configurar uma rede para a comunicação dos contêineres.

1. *docker run -d –net=cluster –name=man1 –ip=192.168.0.2 -v /home/derreck/mysql-cluster.cnf:/etc/mysql-cluster.cnf mysql/mysql-cluster ndb_mgmd*

Para o cluster funcionar é necessário modificar o arquivo padrão *mysql-cluster.cnf* de acordo com quantidade de nodos da arquitetura. Abaixo é demostrado o arquivo utilizado nesse cluster:

```
1. ndbd default
2. NoOfReplicas=2
3. DataMemory=80M
4. IndexMemory=18M
5. ndb_mgmd
6. NodeId=1
7. hostname=192.168.0.2
8. datadir=/var/lib/mysql
9. ndbd
10. NodeId=2
11. hostname=192.168.0.3
12. datadir=/var/lib/mysql
13. mysqld
14. NodeId=4
15. hostname=192.168.0.5
16. mysqld
17. NodeId=5
18. hostname=192.168.0.6
```

Para criar os contêineres de gerenciamento de dados, foi usado os comandos abaixo:

1. *docker run -d –net=cluster –name=ndb1 –ip=192.168.0.3 mysql/mysql-cluster ndbd*
2. *docker run -d –net=cluster –name=ndb2 –ip=192.168.0.4 mysql/mysql-cluster ndbd*

Esses contêineres foi criados em diferentes nodos do cluster para sobrecarregar nenhum nodo.

1. *docker service create*
2. *–name db*
3. *–publish 3306:3306*
4. *–replicas 7*
5. *–env MYSQL_RANDOM_ROOT_PASSWORD=true*
6. *–network cluster*
7. *mysql/mysql-cluster mysqld*

Como mostrado no quadro anterior, o serviço foi criado na mesma rede dos contêineres, seguindo alguma configuração do serviço como: expor a porta 3306 dos contêineres no host Docker; com sete contêineres rodando no cluster; gerando senha do usuário root do SGBD aleatório.

6.2 Resultados Coletados

teste	TE_VM	TE_cluster	VM.cpu	man1.cpu	man2.cpu	man3.cpu	nor1.cpu	nor2.cpu	nor3.cpu	nor4.cpu	man1	man2	man3	nor1	nor2	nor3	nor4
1	12.41	19.88	5.9	29.0	12	14.2	16.5	22.6	28.7	14.1	65.06	54.79	54.84	60.35	83.03	76.87	58.51
2	13.34	22.73	6.4	20.4	11.9	11.0	12.5	25.4	23.8	14.9	57.97	54.49	54.79	60.4	81.59	77.52	58.66
3	13.28	20.82	7.5	17.3	13	12.1	14.0	23.2	23.7	11.2	58.16	54.59	55.14	60.89	81.64	77.62	58.81
4	12.52	16.98	8	17.5	12.8	12.5	14.9	29.6	31.2	12.8	58.96	54.69	55.04	60.15	81.59	77.82	59.16
5	11.73	17.11	6.6	19.6	11.2	9.9	14.0	23.0	24.8	12.9	59.01	54.74	54.94	60	81.74	78.31	59.26
6	9.64	18.88	7.2	19.0	13.4	12.8	12.7	24.7	24.6	11.2	59.11	55.33	54.89	60.55	82.43	78.36	59.31
7	9.55	18.97	6.9	16.9	12.5	11.9	14.7	28.4	25.2	12.3	59.26	55.43	54.99	59.8	82.43	78.46	59.85
8	9.67	19.17	7.5	18.3	11.9	10.3	14.5	23.1	22.9	11.4	59.31	55.48	55.04	59.9	85.41	78.56	59.9
9	10.57	19.07	7.6	18.9	11.1	10.6	13.2	22.3	22.7	11.1	59.5	55.53	55.09	59.7	85.66	78.61	59.95
10	11.69	20.17	6.1	19.5	9.5	12.0	13.9	23.7	24.2	12.4	59.55	55.63	55.14	59.8	85.61	78.66	60
11	11.96	19.82	7.2	18.7	10.9	11.4	11.6	23.0	22.9	9.4	59.75	55.73	55.19	59.65	86.45	79.75	60.1
12	12.41	17.22	7.9	18.4	11.7	11.0	13.7	28.4	22.7	12.5	59.85	55.78	55.29	59.7	86.4	79.75	60.15
13	12.35	16.71	8	9.0	7.5	7.5	8.5	8.9	10.1	9.6	59.9	55.88	55.33	59.75	85.81	77.67	60.2
14	11.91	18.23	6.5	10.2	7.5	8.6	9.6	7.9	8.0	8.9	59.95	55.24	55.43	59.85	86.9	78.96	60.25
15	12.12	16.85	5.9	9.3	8.2	8.2	9.9	8.6	10.3	8.5	60.05	55.33	55.43	59.9	87.44	81.29	60.3
16	12.19	17.17	5.8	9.6	7.5	8.2	9.2	7.9	10.1	9.5	60.3	55.53	55.63	60	87.39	81.49	60.5
17	11.55	17.58	6.6	9.3	7.9	7.9	9.9	9.3	8.8	9.6	60.5	55.58	55.73	59.65	87.25	81.94	60.55
18	11.31	16.39	5.7	44.8	7.5	7.5	8.8	7.9	8.3	8.1	60.5	55.73	55.78	59.7	87.59	82.13	60.65
19	10.09	13.35	6.3	9.3	7.8	7.5	8.2	9.2	8.7	8.2	58.16	55.73	55.83	59.6	87.74	82.43	60.65
20	9.8	12.64	5.2	9.6	8.6	8.2	9.6	9.0	7.7	7.2	58.21	55.83	55.93	59.6	86.95	81.84	60.69
21	9.86	12.4	6.2	9.3	9.5	9.9	9.2	8.9	8.0	8.1	58.31	55.83	55.29	59.65	87.1	81.94	60.79
22	10.4	12.53	7.3	9.6	8.2	8.2	8.2	7.3	9.0	8.6	58.36	55.88	55.33	59.7	87.15	82.03	60.84
23	11.98	12.69	6.2	10.7	8.9	10.1	7.8	9.1	9.4	7.8	58.41	55.29	55.43	59.6	87.25	82.08	60.89
24	11.32	13	7.2	10.7	8.6	8.2	8.1	9.2	8.0	7.9	58.56	55.33	55.43	59.65	87.3	82.23	60.94
25	11.3	13.22	5.3	10.3	8.1	7.8	8.8	10.0	8.0	7.9	58.71	55.38	55.48	59.7	87.3	82.18	60.99
26	11.22	13.07	7.3	19.7	8.5	18.4	59.6	20.1	9.1	71.9	58.66	55.43	55.53	59.75	87.3	82.18	61.04
27	10.87	12.49	7.7	56.2	52.7	65.3	46.6	60.3	9.8	22.4	73	55.63	54.69	62.18	88.64	75.88	55.04
28	10.9	12.5	7.5	9.5	8.6	8.6	9.2	7.3	9.1	7.6	56.87	55.14	56.38	55.58	85.91	78.61	60.99
29	9.49	12.58	6.9	9.9	10.2	8.5	8.8	7.0	8.7	8.3	56.97	55.19	56.43	55.68	87.59	80.15	61.04
30	9.47	12.66	6.8	10.3	8.6	7.6	9.2	6.3	9.2	7.6	57.02	55.24	56.48	55.68	88.39	80.89	61.14

Tabela 15 – Dados coletados do cenário de teste com 7 contêineres

teste	TE_VM	TE_cluster	VM.cpu	man1.cpu	man2.cpu	man3.cpu	nor1.cpu	nor2.cpu	nor3.cpu	nor4.cpu	VM	man1	man2	man3	nor1	nor2	nor3	nor4
1	12.41	12.56	5.9	15.4	25.5	15.4	23.3	22.3	23.9	22.3	7.48	44.76	61.44	51.02	52.31	54.64	51.86	57.37
2	13.34	12.4	6.4	12.5	24.4	17.7	19.8	19.5	20.6	21.5	7.49	44.91	62.63	51.02	52.7	53.75	51.96	56.33
3	13.28	12.44	7.5	12	19.6	14.5	15.8	18	16.5	18.1	7.49	45.01	62.73	51.07	52.26	54.29	52.06	56.18
4	12.52	12.41	8.0	11.3	22.1	13.5	18	18.6	17.1	18.1	7.51	45.11	62.78	51.22	52.41	54.44	52.46	56.28
5	11.73	12.38	6.6	12.1	23	13.4	19.2	16.2	17.3	19.2	7.51	45.26	64.37	51.56	52.41	54.54	52.56	56.87
6	9.64	12.2	7.2	12.6	22.8	16.8	16.9	18.6	17.1	18	7.52	45.36	64.37	51.81	52.41	54.59	52.56	56.92
7	9.55	13.41	6.9	11.9	23.4	14.8	20.1	17.5	17.1	18.3	7.51	45.46	64.37	51.86	52.56	54.69	51.86	57.02
8	9.67	13.86	7.5	12.4	24.6	16	14.6	15.8	15.4	16.9	7.52	45.61	64.57	51.91	52.61	54.74	51.91	56.63
9	10.57	13.99	7.6	13.2	21.1	15.3	16.1	16.2	16.5	15.3	7.5	45.71	67.69	51.96	52.66	54.79	51.76	56.67
10	11.69	13.56	6.1	13	19.1	13.4	15.3	15.9	15.6	16.3	7.49	45.81	67.74	51.96	52.75	54.84	51.86	56.72
11	11.96	11.79	7.2	13.3	20.5	11.9	14.9	16.6	16	15.4	7.49	45.91	68.93	52.01	52.8	54.89	51.91	56.82
12	12.41	11.84	7.9	11.5	18.3	15.2	14.5	16.7	14.7	13.8	7.52	46	70.17	52.01	52.9	54.49	51.76	56.87
13	12.35	11.74	8.0	5.1	7.9	1.4	6.8	8.5	8.2	8.3	7.53	46.1	69.93	51.02	52.95	54.39	51.86	56.92
14	11.91	11.49	6.5	5.1	8.9	1.7	6.9	7.5	9.6	7.2	7.6	46.2	70.12	52.01	53	54.39	51.91	57.02
15	12.12	11.67	5.9	5.5	9.6	1.4	6.8	7.1	6.8	7.7	7.61	45.61	69.63	52.51	53.05	54.49	51.96	57.07
16	12.19	11.85	5.8	5.4	7.5	1.4	7.2	7.1	6.8	7.8	7.61	45.71	69.53	52.41	53.1	54.54	52.06	57.12
17	11.55	12.03	6.6	5.4	8.1	1.4	6.8	7.8	7.6	8.5	7.6	45.76	69.73	52.41	53.25	54.69	52.11	57.27
18	11.31	12.22	5.7	4.8	9	1.4	7.2	7.8	7.2	7.1	7.61	45.81	69.28	52.7	53.35	54.79	52.21	57.32
19	10.09	12.08	6.3	6.4	10.6	1.7	8.2	8.2	8.2	7.8	7.6	45.91	69.93	52.36	53.3	54.84	52.26	57.42
20	9.8	12.83	5.2	6.7	9.5	2	9.6	9.2	8.9	10.2	7.6	45.91	69.93	52.61	53.4	54.94	52.31	56.72
21	9.86	12.86	6.2	6.8	10.3	2.1	8.8	8.2	9.2	9.9	7.61	46	69.93	51.91	53.5	54.99	52.41	56.77
22	10.4	13.2	7.3	7.5	11.3	2	8.8	8.2	9.6	9.2	7.61	46.05	69.53	52.11	52.95	55.04	52.46	56.87
23	11.98	12.36	6.2	7.5	9.5	1.7	9.6	9.2	9.2	10.2	7.61	46.15	69.98	52.26	53	55.14	52.51	56.92
24	11.32	13.09	7.2	6.1	10.9	2	8.9	9.5	8.9	8.8	7.6	46.2	69.68	52.41	53.05	55.19	52.56	57.02
25	11.3	11.96	5.3	6.5	10.5	2.4	8.8	8.9	8.6	9.6	7.61	45.86	70.02	52.46	53.1	55.19	52.61	57.07
26	11.22	11.59	7.3	6.4	10.3	1.7	9.2	10.2	10.2	8.8	7.61	45.86	69.78	52.51	53.2	55.29	52.66	57.12
27	10.87	11.79	7.7	59.1	67.7	71.1	73.7	8.9	8.8	66.8	7.61	46	69.98	52.56	53.25	55.33	52.7	57.22
28	10.9	11.84	7.5	7.1	24.9	2	11.3	56.3	8.9	10.2	7.61	45.96	82.48	56.77	56.38	58.31	53.3	55.73
29	9.49	11.86	6.9	6.3	22.7	2.6	12.6	53.1	8.7	9.7	7.6	45.96	82.53	56.77	56.38	58.31	53.3	55.83
30	9.47	11.71	6.8	6.7	20.5	2.4	11.9	51.8	8.1	9.5	7.6	45.81	82.73	71.71	56.43	58.26	53.45	55.73

Tabela 16 – Dados coletados do cenário de teste com 4 contêineres

teste	VM.TE_VM	TE_cluster	VM.cpu	man1.cpu	man2.cpu	man3.cpu	nor1.cpu	nor2.cpu	nor3.cpu	nor4.cpu	VM	man1	man2	man3	nor1	nor2	nor3	nor4
1	12.41	11.1	5.9	12	18.5	17.7	22.4	10.4	23.5	6.7	7.48	46.8	66.1	65.66	61.24	54.19	58.66	54.69
2	13.34	11.1	6.4	13.1	16	16.5	21.7	9.1	19.4	6	7.49	46.8	66.7	65.76	60.45	54.19	58.26	54.69
3	13.28	11.04	7.5	11.2	15.3	18.5	19.6	9	18.8	6.4	7.49	46.85	67.54	66.6	60.79	54.24	58.11	54.69
4	12.52	11.02	8	13.3	16.2	18	20.8	9.4	18.5	6	7.51	46.9	67.74	67.64	61.29	54.29	58.41	54.69
5	11.73	10.91	6.6	13.6	16.3	17.6	21.9	9	19.1	6	7.51	46.45	67.99	68.34	61.34	54.29	58.56	54.74
6	9.64	10.88	7.2	13.5	16.7	16.7	19.7	9	19.3	6.4	7.52	46.55	68.24	68.44	61.29	54.29	58.41	54.79
7	9.55	11.01	6.9	12	15.2	19.1	19.3	9.4	18.8	6.4	7.51	46.55	68.44	68.49	61.09	54.34	58.66	54.79
8	9.67	11.03	7.5	12.3	19	16.7	20.7	9.7	18.1	6.4	7.52	46.6	68.49	68.68	60.5	54.34	58.66	54.79
9	10.57	11.01	7.6	10.2	15.8	15.7	20	8.1	19.1	6.4	7.5	46.65	68.68	68.88	60.65	54.39	58.26	54.79
10	11.69	10.86	6.1	11.8	17.5	15.5	21.6	9.7	20.8	6.4	7.49	46.7	68.78	68.93	60.3	54.39	57.67	54.84
11	11.96	10.91	7.2	11.5	14.6	18	19.7	11	19.2	6.3	7.49	46.75	68.78	68.93	60.3	54.39	57.72	54.89
12	12.41	11.1	7.9	10.8	20.2	16.8	20.1	9	21.6	6	7.52	46.8	68.83	68.93	60.3	54.39	57.57	54.89
13	12.35	10.91	8	12	17.1	16.9	19.1	10	19.4	7	7.53	46.85	68.88	68.98	60.15	54.44	57.57	54.89
14	11.91	11.01	6.5	11.6	16.2	18.5	20.4	9.7	19.6	6.7	7.6	46.9	68.88	68.98	60.15	54.49	57.62	54.89
15	12.12	11.02	5.9	12.6	17.1	18.7	19.6	10	19	6	7.61	46.9	68.93	69.03	60.2	54.49	57.62	54.94
16	12.19	11.06	5.8	11.6	17.1	21.2	20.4	9.7	19.4	6.4	7.61	47	68.98	69.03	60.2	54.49	57.62	54.94
17	11.55	10.97	6.6	11.2	17.9	17.1	19.1	9.1	19.2	6.3	7.6	47	68.98	69.03	60	54.49	57.62	54.94
18	11.31	11.07	5.7	11.5	16.1	18	20.4	9.4	20.2	5.7	7.61	46.55	69.03	69.08	60.05	54.54	57.67	54.99
19	10.09	11.02	6.3	13	16.7	16.7	19.4	8.4	19.9	6	7.6	46.6	69.03	69.08	60.05	54.54	57.67	54.99
20	9.8	11.06	5.2	12	17.2	15.8	19.6	9.7	18.7	6	7.6	46.65	69.08	69.13	60.1	54.54	57.52	55.04
21	9.86	10.99	6.2	10.5	19.9	16.3	22.1	9	19.8	5.7	7.61	46.7	69.08	69.13	60.1	54.59	57.52	55.04
22	10.4	10.99	7.3	11.3	18.5	17.2	21.2	9.7	19.4	6.4	7.61	46.75	69.03	69.13	60.2	54.59	57.62	55.09
23	11.98	11.03	6.2	11.6	18.2	19.8	21.1	9.4	18.4	6.3	7.61	46.8	69.08	69.13	60.2	54.64	57.67	55.09
24	11.32	10.87	7.2	12.2	14.9	16.5	18.1	10	19.7	7	7.6	46.8	69.08	69.13	60.25	54.64	57.67	55.14
25	11.3	11.04	5.3	11.6	18.4	19	22.3	10.7	21.6	8.7	7.61	46.85	66.75	67.44	60.3	54.64	57.72	55.14
26	11.22	11.09	7.3	11.4	15.7	17.6	23.6	10.8	22.3	7.4	7.61	46.85	67.1	67.89	60.3	54.69	57.77	55.14
27	10.87	10.88	7.7	11.5	20.2	18.1	23.1	10.1	19.7	8	7.61	46.9	67.54	68.54	60.35	54.69	57.57	55.14
28	10.9	11.06	7.5	10.6	16.9	16.6	25.1	10.8	19.4	7.7	7.61	47	67.94	69.03	60.35	54.74	57.62	55.19
29	9.49	11.01	6.9	13.7	18.5	18.5	24.5	9.7	18.8	7	7.6	47	68.54	69.03	60.4	54.74	57.62	55.24
30	9.47	11.07	6.8	10.6	18.6	19	22.9	10.6	18.8	7	7.6	42.18	66.75	69.13	60.25	49.33	57.67	55.24

Tabela 17 – Dados coletados do cenário de teste com 2 contêineres