UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Lucas Ferreira da Silva

IMPLANTAÇÃO DE BANCOS DE DADOS DISTRIBUÍDOS EM UM CLUSTER DE BAIXO CONSUMO

Santa Maria, RS
2022

Lucas Ferreira da Silva

IMPLANTAÇÃO DE BANCOS DE DADOS DISTRIBUÍDOS EM UM CLUSTER DE BAIXO CONSUMO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em Ciência da Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**. Defesa realizada por videoconferência.

ORIENTADOR: Prof. João Vicente Ferreira Lima

Santa Maria, RS
2022

This study was financied in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) – Finance Code 001

Ferreira da Silva, Lucas
Implantação de Bancos de Dados Distribuídos em um Cluster de Baixo Consumo / Lucas Ferreira da Silva.- 2022.
62 p.; 30 cm

Orientador: João Vicente Ferreira Lima
Dissertação (mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Programa de Pós-Graduação em Ciência da Computação , RS, 2022

1. Bancos de Dados Distribuídos 2. Raspberry Pi 3. Docker 4. Computadores de Placa Única 5. NoSQL I. Vicente Ferreira Lima, João II. Título.

Sistema de geração automática de ficha catalográfica da UFSM. Dados fornecidos pelo autor(a). Sob supervisão da Direção da Divisão de Processos Técnicos da Biblioteca Central. Bibliotecária responsável Paula Schoenfeldt Patta CRB 10/1728.

Declaro, LUCAS FERREIRA DA SILVA, para os devidos fins e sob as penas da lei, que a pesquisa constante neste trabalho de conclusão de curso (Dissertação) foi por mim elaborada e que as informações necessárias objeto de consulta em literatura e outras fontes estão devidamente referenciadas. Declaro, ainda, que este trabalho ou parte dele não foi apresentado anteriormente para obtenção de qualquer outro grau acadêmico, estando ciente de que a inveracidade da presente declaração poderá resultar na anulação da titulação pela Universidade, entre outras consequências legais.

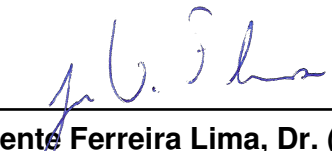
End. Eletr.: lferreira@inf.ufsm.br

Lucas Ferreira da Silva

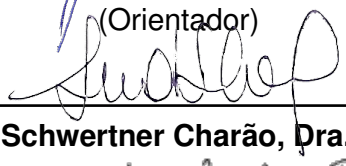
IMPLANTAÇÃO DE BANCOS DE DADOS DISTRIBUÍDOS EM UM CLUSTER DE BAIXO CONSUMO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em Ciência da Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**.

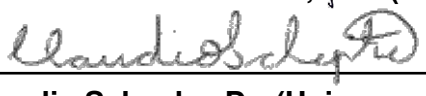
Aprovado em 23 de setembro de 2022:



João Vicente Ferreira Lima, Dr. (UFSM)
(Orientador)



Andrea Schwertner Charão, Dra. (UFSM)



Claudio Schepke, Dr. (Unipampa)

Santa Maria, RS
2022

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Sérgio e Ester, e a meu irmão Joel. Obrigado pelo amor, carinho e apoio incondicionais que sempre me proporcionaram durante minha caminhada pessoal e profissional.

Ao meu orientador João Vicente Ferreira Lima, que sempre esteve a disposição para me auxiliar e guiar, tanto nas decisões relacionadas a este trabalho quanto nas relacionadas ao curso. Agradeço-lhe por ter sido esta pessoa com enorme empatia e que mesmo nos meus piores momentos durante esses últimos anos atípicos, nunca deixou de acreditar no potencial do trabalho nem de minha capacidade concluí-lo. Deixo aqui registrado minha gratidão por sua orientação ímpar e espero que outros alunos tenham o privilégio de serem seus orientandos.

Agradeço a Universidade Federal de Santa Maria, Centro de Tecnologia e Programa de Pós-Graduação em Ciência da Computação pela estrutura, oportunidades e pelo ensino público, gratuito e de qualidade que faz com que o sonho de capacitação de muitas pessoas torne-se uma realidade acessível.

Aos membros da banca de avaliação professor Claudio Schepke e Andrea Schwertner Charão pelas considerações e sugestões engrandecedoras ao trabalho. Um obrigado especial à professora Andrea, que além de parte da banca, tem me acompanhado desde o curso de graduação no papel de orientadora, conselheira, amiga e é uma das minhas maiores referências de pessoa e profissional admirável.

Agradeço também aos colegas do Laboratório de Sistemas de Computação pelos momentos de descontração, debate, troca de conhecimento, auxílio e cafés. Agradecimento especial ao Iago Corrêa, amizade que me acompanha desde a graduação e que compartilhou de vários momentos bons e de dificuldade durante o curso de Mestrado.

Por fim, mas não menos importante, agradeço aos amigos que de alguma forma fizeram parte desta fase de minha vida, seja me incentivando, dando conselhos ou por terem estado presentes de alguma forma. Um muito obrigado aos mais próximos: Ana Veroneze, Andressa G., Cassiano A., Karen R., Márian P., Pedro O., Pedro Q. e Rafael Trindade.

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable

(Leslie Lamport)

RESUMO

IMPLANTAÇÃO DE BANCOS DE DADOS DISTRIBUÍDOS EM UM CLUSTER DE BAIXO CONSUMO

AUTOR: Lucas Ferreira da Silva
ORIENTADOR: João Vicente Ferreira Lima

Os constantes avanços tecnológicos, o crescimento do uso das tecnologias Web, aumento do número de dispositivos móveis e a popularização da IoT (Internet of Things), têm ocasionado um crescimento exponencial de volume de dados nunca antes visto. Este cenário, apesar de positivo sob o ponto de vista tecnológico, traz inúmeros desafios para os centros de processamento e armazenamento de dados, fazendo-se necessária a adoção das ferramentas e tecnologias mais adaptadas a lidar com esse paradigma. Assim, os bancos de dados distribuídos apresentam-se como a solução mais adequada para esse contexto, já que características como escalabilidade horizontal, elasticidade e alta disponibilidade possibilitam que tais tecnologias consigam acompanhar o crescimento vertiginoso do volume e fontes dos dados. Entretanto, da mesma forma que o volume de dados aumenta, aumentam também as demandas de poder computacional, investimento, espaço e consumo energético das infraestruturas como um todo para propiciar a implantação dos bancos de dados distribuídos. Pensando nisso, neste trabalho é explorada a utilização de um cluster de baixo consumo composto por SBCs (Computadores de Placa Única), para a implantação de bancos de dados distribuídos, com o intuito de validar a viabilidade do uso desse tipo de cluster como uma alternativa compacta, barata e com menor consumo energético para composição das infraestruturas dos data centers. Foram utilizados 15 dispositivos Raspberry Pi 3 B para compor o cluster, o qual sustenta uma camada de virtualização por contêineres Docker orquestrados pela ferramenta Docker Swarm. Foram avaliados os desempenhos dos bancos de dados Cassandra, Hbase e PostgreSQL/Citus sobre o cluster de SBCs, sendo utilizadas cargas de trabalho do benchmark YCSB para analisar tempo de execução, latência e throughput em cenários com diferentes fatores de replicação. Os resultados mostram que, no geral, dentre os bancos de dados escolhidos o Cassandra obteve os melhores resultados, além de não demonstrar influência do fator de replicação no seu desempenho. Por outro lado, os resultados para o Hbase e Citus foram fortemente penalizados pelo aumento do fator de replicação. Ademais, a experimentação conduzida evidenciou a capacidade do cluster de baixo consumo de atender as exigências dos sistemas distribuídos utilizados em cenários reais, permitindo ao ambiente distribuído elasticidade e alta disponibilidade.

Palavras-chave: Bancos de Dados Distribuídos, Raspberry Pi, baixo-consumo, Computadores de Placa Única, SBCs, Docker, Docker Swarm, Big Data, NoSQL

ABSTRACT

AN EVALUATION OF RELATIONAL AND NOSQL DISTRIBUTED DATABASES ON A LOW-POWER CLUSTER

AUTHOR: Lucas Ferreira da Silva
ADVISOR: João Vicente Ferreira Lima

The constant technological advances, Web technologies, mobile devices and the popularization of the IoT (Internet of Things), have caused an exponential growth in the volume of data never seen before. This scenario, although positive from a technological point of view, brings many challenges to data processing and storage centers, making necessary the adoption of tools and technologies most adapted to deal with this paradigm. Thus, distributed databases are the most suitable solution for this scenario, since characteristics such as horizontal scalability, elasticity and high availability allow such technologies to keep up with the growing data volume and sources. However, in the same way that the volume of data increases, also increases the demands of computational power, investment, space and energy consumption of the infrastructures as a whole to provide the environment of the distributed databases. In that way, this work explores the use of a low-power cluster composed of SBCs (Single Board Computers), for the implementation of distributed databases, in order to validate the feasibility of using this type of cluster as a compact, cheap and with lower energy consumption alternative for the common data center infrastructures. In that way, this work explores the use of a low-power cluster composed of SBCs (Single Board Computers), for the implementation of distributed databases, in order to validate the feasibility of using this type of cluster as a compact, cheap and with lower energy consumption alternative for the common data center infrastructures. Fifteen Raspberry Pi 3 B devices were used to compose the cluster, which supports a virtualization layer of Docker containers orchestrated by the Docker Swarm tool. The performance of Cassandra, Hbase and PostgreSQL/Citus databases on the SBC cluster was evaluated, using YCSB benchmark workloads to analyze execution time, latency and throughput in scenarios with different replication factors. The results show that, in general, Cassandra outformed the other databases and obtained the best results, showing no influence by the replication factor. The results for Hbase and Citus were heavily penalized by the increase of the replication factor. Furthermore, the results also prove the ability of the low-power cluster to meet the requirements of distributed systems used in real scenarios, allowing to the distributed environment elasticity and high availability.

Keywords: Distributed Databases, Raspberry Pi, low-power, Single Board Computers, Docker, Docker Swarm, Big Data, NoSQL

LISTA DE GRÁFICOS

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

IoT	Internet of Things (Internet das Coisas)
SBC	Single Board Computer (Computador de Placa Única)
SQL	Structured Query Language (Linguagem de Consulta Estruturada)
$RDBMS$	Relational Database Management System (Sistema de Gerenciamento de Banco de Dados Relacional)
$NoSQL$	Not Only SQL (Não Somente SQL)
RAM	Random Access Memory (Memória de Acesso Randômico)
CPU	Central Processing Unit (Unidade Central de Processamento)
JVM	Java Virtual Machine (Máquina Virtual Java)
API	Application Programming Interface (Interface de Programação de Aplicações)
E/S	Entrada/Saída
CLI	Command-line Interface (Interface de linha de comando)
IP	Internet Protocol (Protocolo de Internet)

SUMÁRIO

1 INTRODUÇÃO

Os avanços tecnológicos, a mobilidade dos dispositivos e a popularização de conceitos como a IoT (Internet of Things), tem feito com que o volume de dados gerados esteja crescendo em proporções nunca antes vistas. Essa variedade de dados digitais proveniente das mais diversas fontes aliada a sua grande e crescente quantidade, representa o que é denominado de Big Data, termo que vem ganhando cada vez mais importância tanto no âmbito científico quanto na indústria. Segundo o levantamento realizado pela IDC em 2018 (REINSEL JOHN GANTZ, 2018), espera-se que para o ano de 2025 o volume de dados global chegue a 175 zettabytes, além disso estima-se que 33% desse total serão dados que necessitarão processamento em tempo real.

Nesse cenário, com o aumento do volume de dados também cresce a demanda por ferramentas e tecnologias cada vez mais poderosas e adaptáveis a esse novo paradigma. Dessa forma, começam a surgir diversos problemas nos tradicionais RDBMS, por não terem sido projetados para lidar facilmente com o rápido e exponencial crescimento do volume de dados característico de Big Data. Assim, novas tecnologias de armazenamento como NoSQL (bancos de dados não relacionais) surgem como uma opção mais adaptável ao cenário de Big Data, pois características como a capacidade de lidar com grandes volumes de dados, fácil escalabilidade horizontal e modelo de dados livre de esquema têm motivado a grande adoção desses bancos de dados para tal finalidade. Entretanto, a escalabilidade de armazenamento e processamento desses ambientes exige estruturas de hardware mais robustas e numerosas, ocasionando no aumento do investimento, espaço e consumo energético da infraestrutura como um todo.

Diversos estudos têm avaliado a viabilidade do uso dos SBCs (Computadores de Placa Única, na sigla em inglês) enquanto uma promissora alternativa aos data centers convencionais. Trabalhos como o de (JOHNSTON et al., 2018), apontam que a utilização de dispositivos de baixo consumo como os SBCs podem ser uma opção para minimizar problemas comuns de infraestrutura. Isso porque tais dispositivos permitem que em uma única placa seja possível o encapsulamento de todos os recursos de um computador funcional e com relativo bom poder de processamento que, ao serem interconectados em forma de cluster, podem replicar características dos grandes data centers. Quando comparados com estruturas de processamento de dados tradicionais, (WOLF; JERRAYA; MARTIN, 2008) relata que esses dispositivos são opções compactas, de baixo consumo energético e de baixo custo que entregam uma boa relação entre poder computacional e consumo energético.

Seguindo essa mesma linha, neste trabalho são avaliados três bancos de dados distribuídos executando sobre um cluster composto por 15 dispositivos Raspberry Pi 3 B. Cada um dos três bancos de dados representa uma categoria da classificação proposta

pelo teorema de CAP, sendo escolhidos o Cassandra, o Hbase e o PostgreSQL com Citus para representar as classificações AP, CP e CA, respectivamente. O ambiente distribuído no qual os bancos de dados são executados conta com uma camada de virtualização por contêineres Docker juntamente ao Docker Swarm no papel de orquestrador. Os desempenhos dos três bancos de dados são avaliados através dos resultados obtidos para tempo de execução, latência e throughput, por uma ferramenta de benchmark específica para esse tipo de aplicação, o YCSB.

O restante do trabalho está organizado da seguinte forma: o Capítulo 2 traz algumas explicações fundamentais para o entendimento da arquitetura de cada um dos bancos de dados e funcionamento das tecnologias utilizadas no trabalho, além de apresentar alguns tópicos base para a compreensão dos bancos de dados distribuídos. No Capítulo 3 são elencados alguns trabalhos relacionados ao presente estudo, contextualizando o que vem sendo pesquisado nessa mesma temática ou em assuntos correlatos. A análise comparativa entre os bancos de dados é explicada no Capítulo 4, no qual é detalhado o processo metodológico adotado, a descrição de cada cenário e topologia de teste, os resultados experimentais e a consecutiva discussão dos mesmos. Por fim, no Capítulo 5 são feitas considerações sobre as conclusões obtidas na realização deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo destina-se à apresentação de alguns conceitos referentes aos bancos de dados distribuídos segundo a literatura, além de uma breve explicação das arquiteturas e funcionamento dos três bancos de dados utilizados neste trabalho.

2.1 SISTEMAS DE BANCOS DE DADOS DISTRIBUÍDOS

Um Banco de Dados Distribuído (BDD) pode ser definido como uma arquitetura de banco de dados na qual os dados, logicamente relacionados, são armazenados de forma distribuída sobre um conjunto (cluster) de máquinas, denominadas de nós ou sites. A comunicação e cooperação entre os nós é realizada através de uma rede, podendo esta interconectar máquinas tanto fisicamente próximas por meio de uma rede local, quanto geograficamente distantes por meio redes de longa distância. Os nós que compõem um BDD não compartilham entre si nenhum recurso de hardware (RAM, Disco ou CPU). O Sistema de Gerenciamento de Bancos de Dados Distribuídos (SGBDD) é responsável por coordenar e gerenciar o BDD como um todo, abstraindo dos usuários aspectos tanto de implementação, quanto a forma como os dados são distribuídos. A Ilustração 2.1 sintetiza a topologia geral de um BDD.

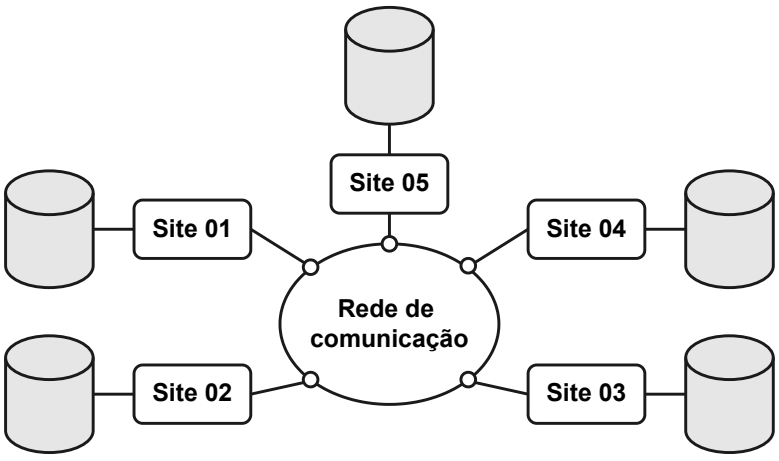


Ilustração 2.1 – Esquema da topologia típica de um BDD.

Fonte: Adaptado de (ELMASRI; NAVATHE, 2015).

De acordo com (SILBERSCHATZ; KORTH; SUDARSHAN, 2020) os Bancos de Dados Distribuídos podem ser classificados como homogêneos ou heterogêneos. Nos BDDs homogêneos, todos nós integrantes do banco de dados são constituídos com as mesmas características de hardware, mesmo Sistema Operacional, executam instâncias do mesmo

SGBDD e compartilham o mesmo (ou compatível) schema[1] de dados, além disso, todos os nós cooperam entre si para o processamento das consultas e transações. Os BDDs heterogêneos, por sua vez, são caracterizados pela presença de Sistema Operacional, hardware e schemas de dados distintos entre os nós do cluster. O SGBDD em execução em cada nó também pode não ser o mesmo, e o processamento cooperativo de transações e consultas tende a ser limitado em relação aos sistemas homogêneos, em virtude da ausência de uma compatibilidade global entre os nós.

2.1.1 Escalabilidade

A escalabilidade pode ser definida como a capacidade de se aumentar o poder de processamento de um sistema conforme a carga de trabalho demandada. Em um BDD, a escalabilidade pode ser dividida em dois tipos: vertical e horizontal.

Na escalabilidade vertical, para aumentar a capacidade do sistema é necessário que os recursos de hardware de um nó individual sejam incrementados e, para isso, geralmente recursos como CPU, RAM e Disco são substituídos por componentes de hardware com maior desempenho. Entretanto, essa estratégia possui a desvantagem de não ser possível melhorar os recursos de uma única máquina infinitamente, pois haverá um momento que o hardware não comportará mais melhorias ou será financeiramente inviável realizá-las, limitando assim a escalabilidade do sistema.

Na escalabilidade horizontal, a estratégia adotada para aumentar a capacidade do sistema consiste em aumentar o número de nós do cluster, possibilitando que máquinas com hardware mais simples sejam utilizadas, já que o processamento do sistema é distribuído no cluster e cada máquina fica encarregada de processar apenas uma porção da carga de trabalho. Essa abordagem é tipicamente uma boa opção para sistemas em que as demandas de processamento e armazenamento crescem constantemente. Em contrapartida, o escalonamento horizontal é mais complexo de ser implementado e requer um projeto minucioso, tendo em vista que decisões de como será realizada a distribuição e processamento dos dados através do cluster não são triviais e afetam diretamente no desempenho geral do sistema.

2.1.2 Teorema de CAP

O Teorema de CAP, ou também Teorema de Brewer, foi proposto no ano de 2000 por Eric Brewer (BREWER, 2000) e posteriormente formalizado em 2002 por (GILBERT;

[1] Organização lógica obedecendo uma estrutura que representa a organização dos componentes e objetos de um banco de dados.

LYNCH, 2002). Esse Teorema afirma que, em um sistema de armazenamento de dados distribuído, não é possível garantir simultaneamente e sempre mais de duas das três seguintes características:

- **Consistência (Consistency - C):** Garantia de que o dado retornado de uma operação de leitura em um nó do cluster seja o mesmo retornado para a operação nos demais nós do cluster, além de representar o dado mais recente e escrito com sucesso na base de dados.
- **Disponibilidade (Availability - A):** Refere-se à garantia de que todo cliente que realiza uma operação receberá uma resposta dentro de um intervalo hábil de tempo, mesmo que nem todos os nós do cluster estejam em funcionamento. Não é assegurado que a resposta para a operação seja condizente com o dado mais recentemente escrito.
- **Tolerância à Partição (Partition Tolerance - P):** A partição, neste caso, refere-se a incapacidade de comunicação entre os nós do sistema distribuído pela rede que os interconecta, seja por interrompimento total ou lentidão. Desta forma, a Tolerância à partição refere-se à garantia de que o sistema distribuído continuará funcionando mesmo que ocorram falhas de comunicação entre os nós do cluster.

Considerando as três garantias citadas, segundo o Teorema é possível classificar os sistemas de bancos de dados distribuídos conforme suas capacidades de assegurar simultaneamente duas das três características CAP em caso de uma falha. Três categorias são possíveis: CP, AP e CA.

Os bancos de dados caracterizados como CP abrem mão da Disponibilidade para assegurar a Consistência e a Tolerância à Partição. Nessa categoria de SBDD quando há uma falha que torne um nó do cluster inacessível, o sistema desativa o nó inacessível/falho das operações do cluster, até que a falha seja corrigida. Tal comportamento faz com que exista a possibilidade de o cluster negar operações de escritas durante a falha, pois é possível que a indisponibilidade de um ou mais nós impeça o correto funcionamento do protocolo de consenso adotado por esses sistemas distribuídos, sacrificando assim a Disponibilidade. Hbase (HBASE, 2021a) e Redis (Redis Ltd., 2022) são exemplos de bancos de dados classificados como CP.

Os bancos de dados do tipo AP possuem como política o sacrifício da consistência em prol da garantia de Disponibilidade e Tolerância à Partição. Nestes sistemas, quando ocorre uma partição, todos os nós do cluster ainda permanecem disponíveis, entretanto, como a comunicação entre os nós necessária para a sincronização dos dados do cluster está afetada, os nós que estão inacessíveis não podem ser sincronizados com os demais, possibilitando então que um nó não sincronizado possua um dado não consistente. Todavia, os SBDDs caracterizados como AP adotam o conceito de Consistência Eventual (do

inglês, Eventual-Consistency) que possibilita que o sistema continue permitindo operações de escrita (mesmo com um ou mais nós falhos) e postergando a sincronização dos dados com os nós inacessíveis para quando eles ficarem disponíveis novamente (BREWER, 2012). Essa janela de tempo entre as novas escritas e a sincronização dos dados com a parte do cluster indisponível é o período no qual o sistema não consegue garantir a Consistência. Cassandra (APACHE, 2022) e DynamoDB (AWS, 2022) são exemplos de bancos de dados classificados como AP.

Os bancos de dados CA por sua vez priorizam a alta Disponibilidade e forte Consistência em detrimento da Tolerância à Partição. Teoricamente, tais sistemas possuem a peculiaridade de que em caso de falha em algum dos nós do cluster todo o sistema fica indisponível até que a falha seja resolvida. Isso ocorre porque os bancos de dados classificados com CA não possuem mecanismos eficientes de garantia de resiliência em caso de indisponibilidade de um dos nós do cluster, ou até mesmo são arquiteturalmente limitados para tal. Conforme mencionado por (IBM Cloud Education, 2019), embora pareça um pouco contraditório a existência de um SBDD não tolerante a partição, ainda assim é possível a estruturação de uma arquitetura de banco de dados distribuída e que garanta Consistência e Disponibilidade com o relaxamento da Tolerância à Partição, a exemplo, podem ser citados os bancos de dados relacionais que garantem muito bem as características CA e, com a utilização de técnicas de replicação de dados, podem ser distribuídos por vários nós em um cluster. PostgreSQL (GROUP, 2022) e MySQL (Oracle Corporation, 2022), ambos SGBDs relacionais, são exemplos de bancos de dados que podem ser classificados como CA, se operando em modo distribuído.

2.2 CASSANDRA

Cassandra (LAKSHMAN; MALIK, 2010) é um banco de dados não relacional, distribuído e de código aberto, que tem por base para seu sistema de distribuição de dados o Dynamo da Amazon (DECANDIA et al., 2007) e como modelo de dados o Bigtable da Google (CHANG et al., 2008).

O Cassandra possui uma arquitetura distribuída descentralizada, pois não estabelece uma hierarquia entre os nós de que fazem parte do cluster, permitindo assim que cada nó possa atender individualmente qualquer requisição para a base de dados evitando a existência de um ponto único de falha.

A consistência de dados do Cassandra pode ser ajustada a partir da seleção de um dos níveis de consistência disponíveis. A partir dessa configuração, realizada no cliente a cada operação, que se define quais réplicas que participarão de uma operação de leitura ou escrita, sendo necessário que tais operações satisfaçam o nível de consistência escolhido para serem consideradas bem-sucedidas.

Além do nível de consistência, pode ser também especificado o fator de replicação dos dados. O fator de replicação nada mais é que o número de réplicas de um determinado dado que serão distribuídas entre os nós do cluster. Essa configuração permite que o nível de tolerância a falhas e a resiliência da base de dados sejam personalizados conforme necessário.

A comunicação entre os nós do cluster Cassandra é realizada por meio do protocolo Gossip, que utiliza uma comunicação ponto-a-ponto a qual permite que os nós fiquem trocando informações periodicamente sobre seu estado e o estado dos demais nós dentro do cluster, garantindo assim que possíveis falhas sejam identificadas em tempo real (Apache Cassandra, 2016b).

O particionamento dos dados entre os nós do cluster é realizado dinamicamente por meio de um algoritmo de hash consistente, que faz com que cada nó seja responsável por uma porção dos dados, de acordo com o intervalo estabelecido por um token gerado para o nó. Caso haja um aumento no volume de dados ou um novo nó ingresse no cluster, automaticamente os valores dos tokens serão recalculados para que cada nó possua uma porção de dados proporcional a dos demais. Na Figura 2.2 é apresentado um exemplo de distribuição de um intervalo de 0 a 255 tokens em um cluster de 4 nós.

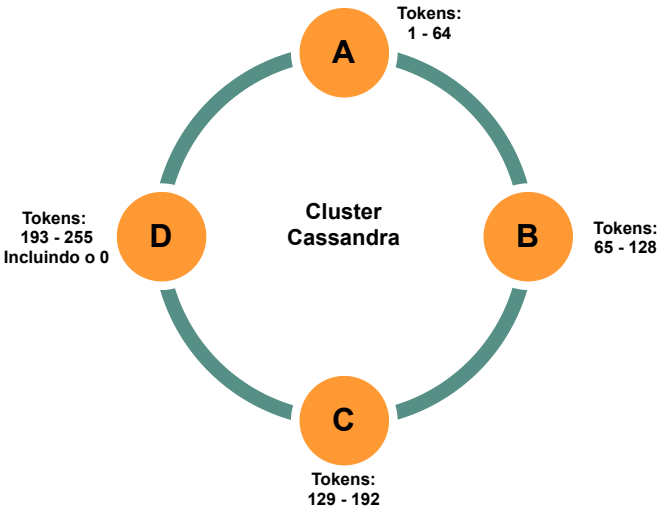


Ilustração 2.2 – Distribuição de tokens em um *cluster* Cassandra.

Fonte: Adaptado de (Apache Cassandra, 2016a).

2.3 POSTGRESQL

O PostgreSQL é um sistema de gerenciamento de banco de dados (SGBD) de código aberto que utiliza e estende a linguagem SQL. Criado em meados de 1986, o PostgreSQL surgiu na Universidade da Califórnia em Berkeley (Estados Unidos) como parte

do projeto POSTGRES (Stonebraker; Rowe; Hirohama, 1990), o qual foi renomeado anos mais tarde para PostgreSQL no intuito de ressaltar o uso da linguagem SQL.

O PostgreSQL é um banco de dados do tipo cliente-servidor e internamente é composto por uma arquitetura de multi-processos. Segundo a documentação oficial do PostgreSQL (POSTGRESQL, 2021), o conjunto dos múltiplos processos que executam cooperativamente para o gerenciamento do banco de dados é usualmente denominado de postgres server (Backend) que, em conjunto com um cliente (Frontend), constituem uma sessão do PostgreSQL. A ilustração 2.3 esquematiza de forma resumida como o PostgreSQL está organizado arquiteturalmente.

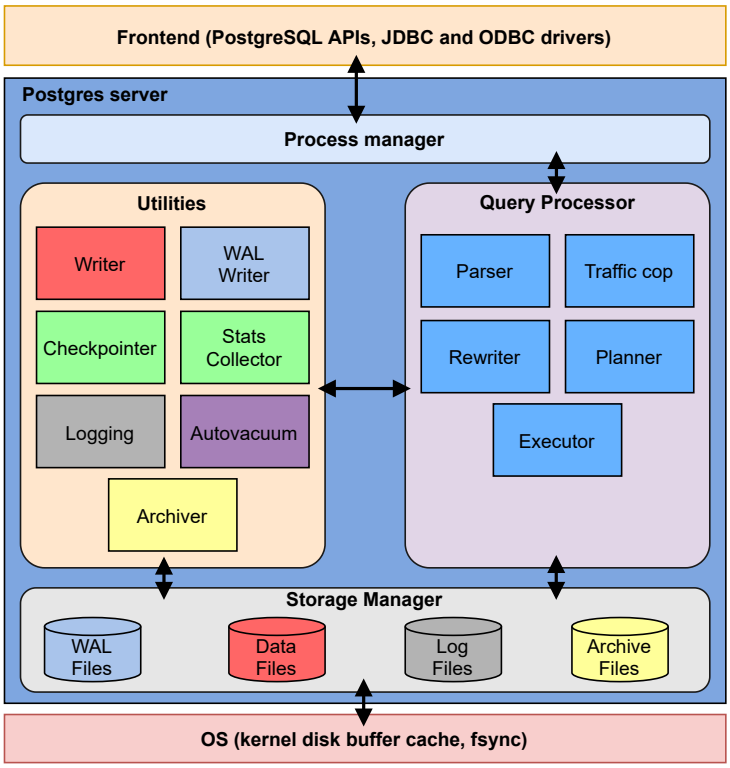


Ilustração 2.3 – Organização arquitetural do PostgreSQL.

Fonte: Adaptado de (JUBA; VOLKOV, 2019).

O postgres server é responsável por lidar com as conexões ao banco de dados realizadas pelos clientes, gerenciar os arquivos do banco de dados e executar demais ações que estão por trás da interação entre os clientes e o servidor. A comunicação entre os clientes e o servidor é realizada por meio do protocolo TCP/IP ou por meio de Sockets Linux. A cada nova conexão de um cliente, é realizado um fork do processo principal do servidor postgres e somente o novo processo passa a interagir com a conexão do cliente, sem a intervenção do processo principal que fica apenas na espera por novas conexões. Essa estratégia permite que o PostgreSQL consiga atender múltiplas conexões concorrentes e de forma simultânea.

Como pode ser observado na ilustração 2.3, o servidor postgres é composto por

quatro subsistemas principais:

1. **Process Manager (Gerenciador de processos):** o Process Manager é iniciado juntamente com o início da execução do postgres e é responsável por tarefas importantes como recuperação (recovery), inicialização das estruturas de dados compartilhadas e dar início aos demais processos do postgres server. O Process Manager ainda é responsável por gerenciar a criação (forks) e finalização de processos atrelados às conexões realizadas por clientes, além de poder gerenciar processos de tarefas em segundo plano como os serviços de log.

2. **Query Processor (Processador de consulta):** o Query Processor é o subsistema que realiza o processamento das consultas realizadas ao banco de dados. Cada consulta passa por várias etapas no Query Processor durante sua execução: verificação de sintaxe, criação da árvore de consulta, reescrita da árvore de consulta com as devidas transformações, criação do plano de consulta otimizado e execução do plano de consulta que retorna os resultados.

3. **Utilities (Utilitários):** correspondem ao grupo de processos com finalidades específicas mas que atuam de forma cooperativa para prover as funções do postgres server.

4. **Storage Manager (Gerenciador de armazenamento):** o Storage Manager é o encarregado de gerenciar a memória cache do postgres server, os buffers de disco e a devida alocação dos dados.

2.3.1 Escalabilidade com PostgreSQL

Em sua implementação padrão, o PostgreSQL possui recursos apenas para prover escalabilidade vertical e, portanto, o desempenho do banco de dados está atrelado à quantidade de recursos de hardware disponíveis na máquina na qual o SGBD está executando. Nesse modelo de escalabilidade, para a obtenção de um melhor desempenho o PostgreSQL permite o uso de algumas técnicas de replicação de dados, que consistem em ter cópias de um mesmo dado ou relação em instâncias distintas do PostgreSQL. Essas técnicas utilizadas podem ser classificadas em dois grupos: replicação física e replicação lógica.

A replicação física é utilizada para manter uma cópia completa de todos os dados de uma única instância PostgreSQL. Essa instância que é copiada é chamada de primary e a a instância com sua cópia é chamada de standby, sendo essa última geralmente alocada em uma máquina física distinta. Esse tipo de replicação opera diretamente com os arquivos

de dados, fazendo com que a instância standby possua uma cópia exata, a nível de bytes, da instância primary. Desta forma, os resultados de alterações de qualquer magnitude na instância primary resultam em uma instância standby exatamente igual.

Entretanto, a replicação física possui algumas desvantagens mesmo que garanta a consistência e disponibilidade do sistema, pois ao operar a nível de sistema de arquivos, mesmo operações que não afetem o conteúdo dos dados armazenados resultam na ação do sistema de replicação, o que é muitas vezes desnecessário quando o objetivo da replicação for apenas garantir disponibilidade e a consistência do banco de dados. Nessas situações a replicação lógica passa a ser uma alternativa.

A replicação lógica, diferentemente da replicação física, opera no nível das tabelas do banco de dados, sendo que ao invés de propagar para outra instância o resultado das operações, a replicação lógica faz com que a consulta executada na instância principal seja também executada nas demais instâncias. A interação entre as instâncias na replicação lógica também é diferente, pois não há instâncias primary e standby e sim instâncias publisher, que realizam o envio dos dados de replicação, e as instâncias denominadas subscriber que apenas recebem tais dados. Esse sistema de funcionamento permite que uma mesma instância possa ao mesmo tempo ser publisher para determinadas tabelas e subscriber para outras (SCHöNIG, 2015).

Apesar das técnicas de replicação física e lógica, conforme o banco de dados cresce, ele passa a demandar cada vez mais recursos da máquina na qual a instância do PostgreSQL está executando e mesmo com o aprimoramento dos recursos de hardware, chega um momento que escalar verticalmente o sistema se torna inviável e é preciso adotar outras estratégias. Uma solução para esse cenário é escalar o banco de dados horizontalmente, porém, em sua implementação padrão, o PostgreSQL não é um banco de dados distribuído e implantar esse tipo de escalabilidade não é algo trivial, apesar de possível.

2.3.2 Citus

Citus (CUBUKCU et al., 2021) é uma extensão para o PostgreSQL que permite a distribuição dos dados e consultas através de vários nós de um cluster, viabilizando a escalabilidade horizontal. Por ser uma extensão para o PostgreSQL, não uma ramificação do projeto principal, Citus permite que todos os recursos presentes no PostgreSQL e seu ecossistema sejam utilizados normalmente e em suas versões mais recentes.

O PostgreSQL é um banco de dados relacional catalog-driven (em tradução livre, orientado a catálogo), o que significa que o sistema baseia todo seu funcionamento nos dados contidos nos catálogos de sistema, que são, de forma resumida, uma base de dados contendo metadados sobre tabelas, colunas, bases de dados, etc. Diferentemente de

outros bancos de dados, o PostgreSQL registra em seus catálogos diversas informações além das comumente catalogadas, informações como: metadados de tipos de dados, funções e métodos de acesso. Além disso, tais catálogos são acessíveis para leitura e escrita, fazendo com que os usuários possam extender e/ou modificar livremente o funcionamento do PostgreSQL, já que sua operação é catalog-driven. Desta forma, a implementação do Citus utiliza-se dessa flexibilidade e facilidade de extensão do PostgreSQL para, por meio da utilização e/ou modificação de recursos como técnicas de sharding, motor de consulta distribuído e sistema de transação distribuída, permitir a implementação de uma extensão que integre ao PostgreSQL os requisitos e as funcionalidades de um banco de dados distribuído.

Um cluster Citus é formado por dois tipos de nós: trabalhadores (workers) e coordenadores (coordinators). Geralmente, há no mínimo um ou vários nós coordenadores e nenhum ou vários nós trabalhadores, todos executando instâncias do PostgreSQL. Os nós trabalhadores são responsáveis pelo armazenamento dos shards das tabelas distribuídas e, por consequência, da execução das consultas distribuídas referentes ao segmento de dados que armazenam.

Os nós coordenadores são responsáveis por armazenar os metadados sobre o cluster e distribuição dos dados, além de serem os nós nos quais os clientes se conectam para realizar consultas ao banco de dados. Entretanto, os nós coordenadores podem concomitantemente desempenhar o papel de armazenando dados de um worker, o que possibilita a existência de um cluster Citus de apenas um único nó, sendo nesse caso, o nó autônomo quanto ao gerenciamento do cluster e ao armazenamento dos dados. Na Ilustração 2.4 é apresentado um exemplo da topologia de um cluster Citus.

Citus adiciona ao PostgreSQL um novo tipo de tabela: as tabelas distribuídas. As tabelas distribuídas foram criadas com o intuito de abstrair as operações envolvidas na ditribuição dos dados permitindo que os usuários acessem os dados de uma tabela como se fosse uma tabela local. Nas tabelas distribuídas, os dados são particionados por meio de um algoritmo de hash sobre uma coluna especificada como a determinante da distribuição, resultando em shards que serão alocados nos nós do cluster utilizando a estratégia de round-robin. Cada nó do cluster pode conter vários shards, sendo que preza-se (sempre que possível) por manter shards com dados contíguos em um mesmo nó. Por se utilizar uma estratégia de round-robin para a alocação dos shards, obtêm-se uma distribuição relativamente equilibrada do número de shards por nó, entretanto, eventualmente tal distribuição pode não ser mais homogênea, como no caso da adição de um novo nó ao cluster após já existirem shards distribuídos. Nesses casos, Citus possibilita o rebalanceamento dos shards existentes através de uma função específica, a qual pode executar um rebalanceamento padrão, ou aplicar alguma política personalizada pelo usuário (CitusData, 2022).

Para realizar uma consulta à base de dados distribuída, os usuários se conectam a um nó coordenador e interagem com o banco de dados da mesma forma como é feito

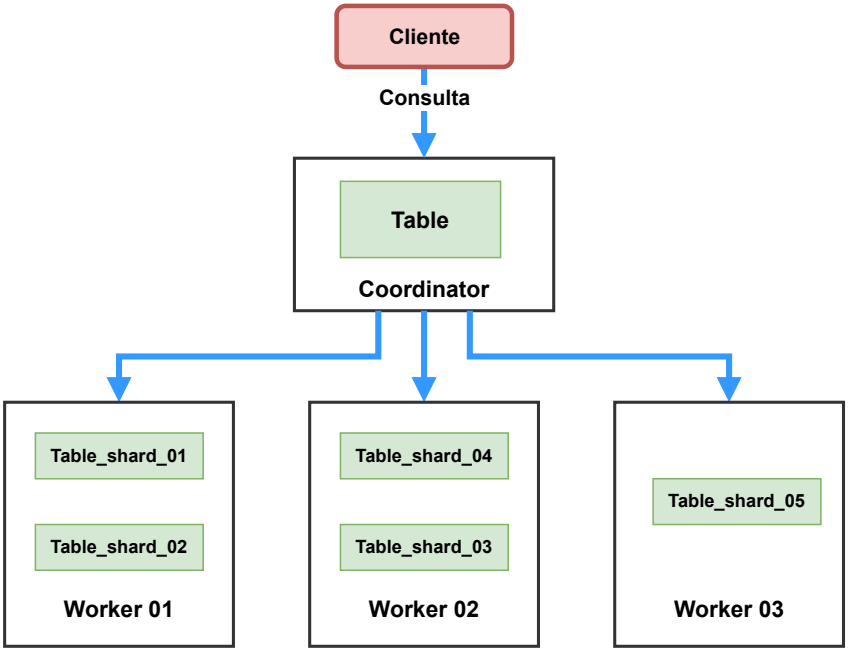


Ilustração 2.4 – Topologia distribuída de um cluster Citus.

Fonte: Adaptado de (CitusData, 2022)

em um banco de dados de um único servidor. O nó coordenador do cluster Citus abstrai todas as tarefas de acesso e manipulação dos dados distribuídos e entrega ao usuário final apenas o resultado da operação.

Durante a execução de uma consulta, primeiramente um planejador de consulta distribuída é executado no nó coordenador, esse planejador é responsável por analisar a consulta e criar um plano otimizado para sua execução. No Citus, existem algoritmos de planejamento específicos para cada classe de consulta, sendo selecionado o que se encaixa melhor nas características da consulta em questão (CUBUKCU et al., 2021).

O resultado da execução do planejador de consulta distribuída é utilizado por um executor de consultas distribuídas. Nos casos em que a consulta afetará apenas os dados de um único worker, o executor delega a execução diretamente para o nó em questão, onde o plano de consulta é então executado. Já nas consultas que afetam os shards presentes em mais de um nó worker, o executor de consulta distribuída propaga para cada um dos nós workers o seu respectivo plano de consulta. Nos casos em que a consulta envolve o retorno de dados que estão em shards de mais de um nó, a execução do plano de consulta em cada worker resulta em um subconjunto de dados. Esses subconjuntos são agregados por um procedimento específico do executor de consulta distribuída, que resulta em um único conjunto de dados o qual corresponde ao resultado final da consulta distribuída.

2.4 HBASE

O Apache HBase é um banco de dados NoSQL orientado a coluna e de código aberto projetado para suportar grandes volumes de dados, cujo desenvolvimento é baseado no Google BigTable (CHANG et al., 2008). O HBase é nativamente distribuído e utiliza-se do Hadoop Distributed File System (HDFS) (SHVACHKO et al., 2010) para prover recursos de armazenamento distribuído semelhantes ao do BigTable executado sobre o sistema de arquivos do Google. O uso do HDFS permite que o HBase integre recursos do Hadoop (HADOOP, 2006), como o MapReduce, para o processamento e manipulação dos dados.

Por ser linearmente escalável, o HBase pode atender grandes demandas de armazenamento, possibilitando que dados de diferentes estruturas e esquemas sejam armazenados com facilidade e de maneira tolerante a falhas. Além disso, o HBase provê que operações de leitura e escrita em grandes volumes de dados possam ser realizadas em tempo real.

A arquitetura do HBase é baseada no modelo mestre/escravo, na qual há o nó denominado HMaster desempenhando o papel de mestre e vários nós HRegionServers com o papel de escravos. Cada HRegionServer pode ter um ou mais Regions, que nada mais são que segmentos de uma tabela de dados distribuída.

O HMaster é responsável pelas atividades administrativas do cluster, como monitorar as instâncias de HRegionServers, atribuir Regions aos HRegionServers e realizar o balanceamento de carga do cluster. Já os HRegionServers, destinam-se ao armazenamento e gerenciamento dos Regions, lidar com as operações de leitura e escrita e responder às conexões diretas com clientes do banco de dados. Cada HRegionServer é composto pelos componentes:

- **Hfiles:** Formato de arquivo utilizado pelo HBase que armazena dados no formato chave/valor.
- **WAL (Write-Ahead Log):** é um arquivo presente no sistema de arquivos distribuído utilizado para armazenar dados que ainda não foram definitivamente registrados na base de dados. Os arquivos WAL podem ser também utilizados na recuperação de dados em caso de falhas.
- **BlockCache:** é uma cache destinada a manter em memória os blocos de dados lidos com mais frequência e mais recentemente dos Hfiles, com o intuito de reduzir os acessos ao disco.
- **MemStore:** é um buffer utilizado como cache de escrita que permite que o HBase mantenha os dados em memória antes de gravá-los permanentemente em disco.

Para o gerenciamento de estado do cluster, o HBase utiliza o Zookeeper (ZOOKEEPER, 2010) como sistema de coordenação distribuído. O Zookeeper é responsável por estabelecer a comunicação entre clientes e HRegionServers, manter informações referentes a configuração e topologia do cluster, além de monitorar o estado dos nós do cluster, gerando notificações em caso de falhas. Por padrão o HBase possui o próprio Zookeeper integrado, cujo processo ganha o nome de HQuorumPeer. Entretanto, é possível criar e gerenciar um cluster Zookeeper à parte e especificar para o HBase que sejam utilizadas as instâncias externas do Zookeeper ao invés da implementação nativa.

O HBase possui dois modos de operação: Standalone e Distributed. No modo Standalone (modo padrão), o HBase utiliza o sistema de arquivos local da máquina e a utilização do HDFS torna-se dispensável. Nesse modo, o HBase não opera de forma distribuída, sendo que todos os serviços (HMaster, HRegionServer e Zookeeper) são executados na mesma JVM de uma mesma máquina física. Esse modo geralmente é utilizado apenas para testes e desenvolvimento, sendo desaconselhado para ambientes de produção (HBASE, 2021b).

O modo Distributed pode ser subdividido ainda em dois outros modos de operação: Pseudo-distributed e Fully-distributed. Em Pseudo-distributed o sistema de arquivos utilizado pode ser tanto o local quanto o HDFS e os serviços do HBase, assim como no modo Standalone, irão executar na mesma máquina física, porém em instâncias de JVM distintas. O modo Fully-distributed por sua vez, é o modo que de fato é distribuído e o recomendado para utilização em ambientes de produção, nele os serviços do HBase são executados em nós distintos do cluster e, por esse motivo, a utilização do HDFS como sistema de arquivos torna-se obrigatória (HBASE, 2021a). A Ilustração 2.5 exemplifica como é a organização arquitetural de um cluster HBase em modo Fully-distributed.

Quando operando em modo Fully-distributed, tipicamente os HRegioServers executam nos nós do cluster que estão executando instâncias Datanode do HDFS (responsáveis por armazenamento e particionamento dos dados), o que possibilita o armazenamento dos Hfiles de forma distribuída sobre os HDFS. O HMaster, por ser responsável apenas por tarefas de gerenciamento e não de persistência de dados, é comumente executado no mesmo nó do cluster ao qual há uma instância do Namenode do HDFS em execução, já que o Namenode também é responsável apenas por tarefas de manutenção e gerenciamento do sistema de arquivos distribuído.

2.5 RASPBERRY PI

Raspberry Pi é um computador de placa única que, diferentemente dos computadores comuns, incorpora em uma única placa de circuito todas as funcionalidades necessárias e básicas de um computador funcional. Desenvolvido no Reino Unido pela Fundação

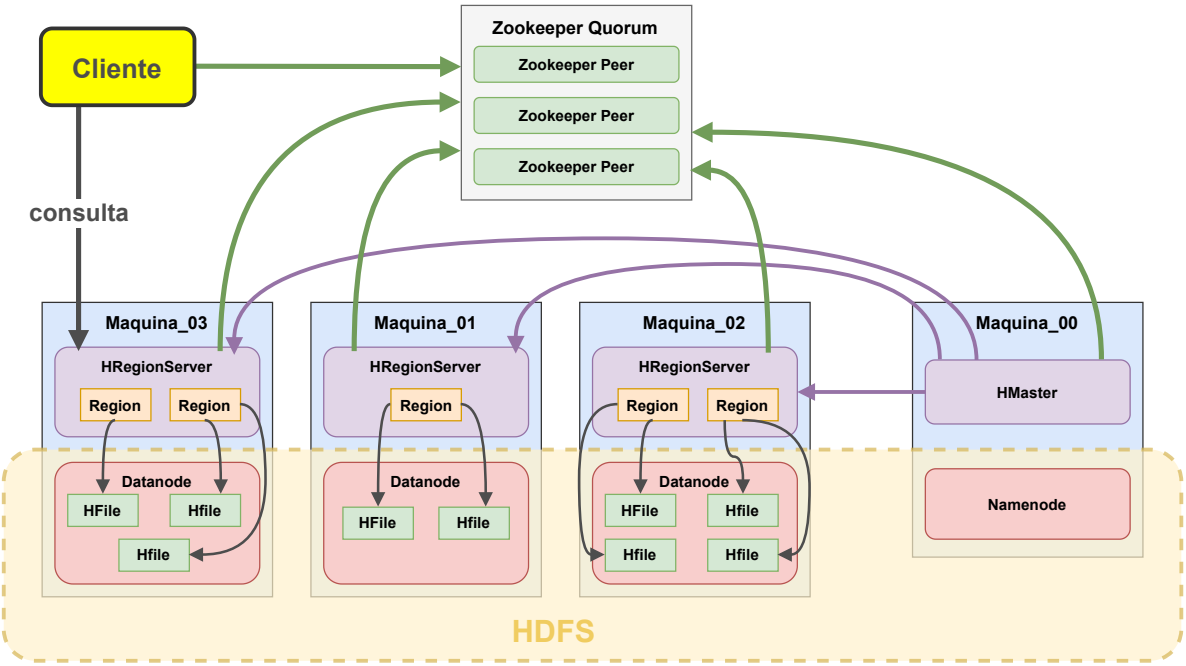


Ilustração 2.5 – Organização arquitetural de um cluster HBase.

Fonte: Adaptado de (IBM, 2014).

Raspberry Pi, começou a se popularizar em 2012 no ecossistema open source com o objetivo de ser um recurso de auxílio no âmbito do ensino de computação, porém, logo se popularizou entre entusiastas do mundo da eletrônica e do hardware por seus inúmeros recursos e seu baixo preço.

Em sua primeira versão o Raspberry Pi contava com um processador de 1 único núcleo de 700MHz e apenas 256MB de RAM, já a sua versão mais recente, o Raspberry PI 4, conta com um processador de quatro núcleos com frequência de 1.5GHz e opções de capacidade de RAM que variam entre 2GB, 4GB ou 8GB (Raspberry Pi Foundation, 2020). O sistema operacional oficialmente adotado para o Raspberry é o Raspbian (recentemente renomeado para Raspberry Pi OS), que é uma versão modificada do Linux Debian destinado às arquiteturas de processador ARMv7. Além das funcionalidades de um computador comum, o dispositivo conta com GPIOs (do inglês: General Purpose Input/Output) que possibilitam a interface com outros periféricos ou hardware por meio de portas programáveis, recursos esse que possibilitou a popularização do Raspberry nos projetos de IoT (NUTTALL, 2019).

Atualmente o dispositivo é utilizado no mundo todo para diversas finalidades sendo algumas delas: ensino e aprendizado de computação, programação e prototipação de eletrônica e hardware, e na automação residencial e industrial.

2.6 DOCKER

Docker é uma plataforma de código aberto que permite a criação de imagens personalizadas que servem como base de definição para contêineres. Segundo (DOCKER, 2021a), um contêiner é uma unidade padrão de software que empacota o código e todas as suas dependências para que o aplicativo seja executado de maneira rápida e confiável independente do ambiente. Já uma imagem Docker, pode ser designada como sendo um empacotamento de instruções sequencialmente dispostas em um arquivo de template (o Dockerfile), tais camadas são denominadas como as “layers” de uma imagem Docker e, quando executadas pelo driver Docker, dão origem a instâncias de software que nada mais são que os próprios contêineres.

Diferentemente das máquinas virtuais, que baseiam-se na abstração do hardware da máquina hospedeira por meio de um hypervisor, a virtualização por meio de contêineres, ou conteinerização, é realizada a nível de sistema operacional, permitindo assim que os recursos da máquina física sejam particionados, criando vários espaços de usuário isolados (os contêineres), executando sobre o mesmo kernel do sistema operacional da máquina hospedeira do sistema (Ilustração 2.6). Dentre as vantagens dessa abordagem, destaca-se que não há a sobrecarga de execução, pois esse tipo de virtualização em nível de sistema operacional não necessita da abstração do hardware da máquina, dispensando a existência de um hypervisor.

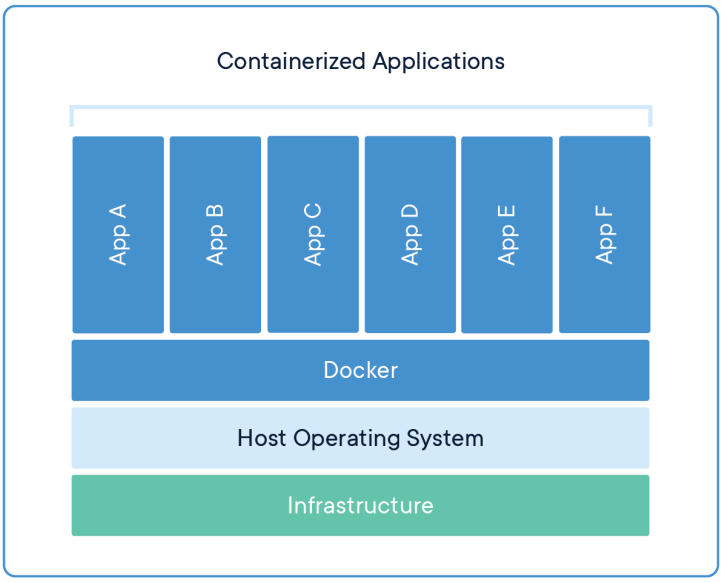


Ilustração 2.6 – Virtualização por meio de contêineres Docker.

Fonte: (DOCKER, 2021a).

Inicialmente, driver de execução do Docker era o LXC (Linux Containers), porém, com o passar do tempo e evolução do projeto, a equipe por trás da plataforma implementou seu próprio driver (MORABITO; KJäLLMAN; KOMU, 2015). Além disso, para a disponibili-

zação das imagens, a plataforma Docker conta com o serviço Docker Hub, que possibilita que os usuários do Docker possam compartilhar suas imagens de forma fácil, para que as mesmas estejam acessíveis por outros usuários, facilitando a replicação e implantação de um determinado ambiente virtual em ambientes físicos distintos.

2.6.1 Docker Swarm

O Docker Swarm é uma ferramenta de orquestração de contêineres incorporada nativamente na plataforma Docker através da biblioteca SwarmKit. O Docker Swarm possibilita a criação e gerenciamento de um cluster composto por máquinas executando o driver Docker, os quais interagem entre si por meio da API Docker (DOCKER, 2021b). Cada uma dessas máquinas, ou nós Docker, pode desempenhar o papel de gerente, trabalhador ou executar as duas funções. O nó gerente é encarregado de desempenhar as funções de orquestração e gerenciamento do cluster, sendo ele o responsável pela distribuição das unidades de trabalho (denominadas "tasks") entre os demais nós do cluster. Nós trabalhadores em contrapartida, não possuem responsabilidades atreladas à manutenção do cluster, sendo a execução das tasks sua única função no cluster. A Ilustração 2.7 exemplifica como um cluster swarm pode ser estruturado.

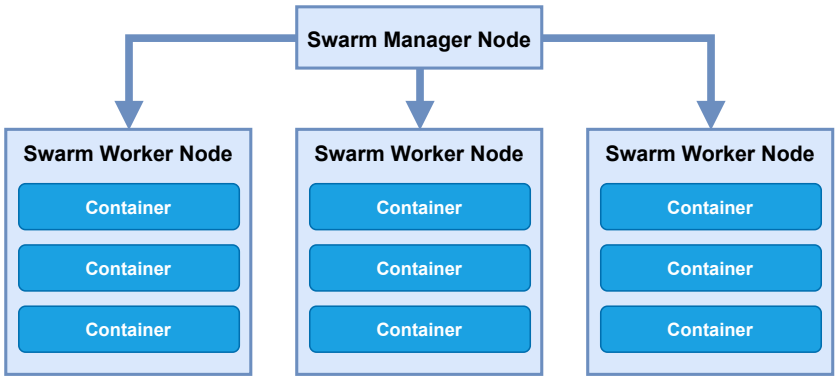


Ilustração 2.7 – Estrutura de um cluster Docker Swarm.

Fonte: Adaptado de (DEB, 2021)

A definição das tasks a serem executadas nos nós do cluster Swarm é chamada de serviço. Ao estruturar um serviço para um cluster Swarm são especificados detalhes de cada uma das tasks que serão executadas, sendo possível definir a imagem Docker a ser utilizada, quais comandos serão executados, configurações de rede, definição de volumes, entre outras diversas configurações. É possível definir dois possíveis modos de operação para um serviço Swarm: o modo "replicado" e o modo "global". No modo replicado, o gerenciador do cluster Swarm distribui um número específico de tasks entre os nós do cluster, seguindo o valor de réplicas definido para o serviço em questão. Já nos serviços

no modo global, o gerenciador do cluster Swarm distribui uma tarefa do serviço para cada um dos nós parte do cluster e disponíveis.

3 TRABALHOS RELACIONADOS

Bancos de dados distribuídos, mais especificamente os bancos de dados NoSQL têm sido alvo de diversos trabalhos com o intuito de explorar suas potencialidades frente aos bancos de dados relacionais, ou até mesmo objetivando avaliar características de desempenho. Alguns trabalhos como os de (KIM et al., 2020) apresentam técnicas e recomendações que podem ser utilizadas na migração de bancos de dados relacionais para bancos de dados NoSQL, comparando o banco de dados MySQL com o Hbase e utilizando o Apache Phoenix como uma camada para viabilização do uso de consultas SQL no Hbase. Outros estudos, mais detalhados, como o de (DAVOUDIAN; CHEN; LIU, 2018) trazem uma análise aprofundada sobre os bancos de dados não relacionais levando em conta seus aspectos arquiteturais de modelo de dados, modelo de consistência, particionamento e teorema de CAP. Pelo fato de os bancos de dados NoSQL trazerem consigo características como alta escalabilidade e capacidade de armazenamento de grandes volumes de dados, eles ganharam espaço de destaque no âmbito de Big Data, motivando pesquisas como a de (HENDAWI et al., 2019), no qual a escalabilidade horizontal e vertical são utilizadas na comparação do Cassandra, MongoDB e Hbase para persistência de volumes de dados do contexto de IoT. Os resultados obtidos pelos autores demonstraram que o Cassandra se destacou dentre os demais bancos de dados com melhores resultados para o cenário distribuído. Seguindo essa linha, vários trabalhos realizam a análise comparativa de bancos de dados por meio de ferramentas de benchmark específicas para tal finalidade, possibilitando que se estabeleça um critério comparativo comum e replicável das experimentações entre pesquisas distintas. Atualmente, algumas das ferramentas de benchmark mais citadas e adotadas são o NDBench (PAPAPANAGIOTOU; CHELLA, 2018) e YCSB (COOPER et al., 2010), utilizado nos trabalhos de (ABRAMOVA; BERNARDINO; FURTADO, 2014; Swaminathan; Elmasri, 2016; ABRAMOVA; BERNARDINO, 2013).

O uso de clusters formados por dispositivos de baixo consumo energético tem motivado um crescente número de trabalhos no intuito de explorar as potencialidades desse ambiente distribuído para objetivos computacionais mais específicos. (ASHARI; RIASETIAWAN, 2015) demonstram por meio da realização de benchmarks de CPU em clusters de dispositivos Raspberry Pi que, mesmo sendo dispositivos com limitações de recursos em comparação com arquiteturas mais rápidas, esses clusters são capazes de fornecer uma boa relação de poder computacional e consumo energético. Já o trabalho de (PRIYAMBODO; LISAN; RIASETIAWAN, 2018) além da execução de benchmarks para avaliar o desempenho de CPU, demonstram que é possível aumentar a relação entre desempenho e consumo energético por meio de técnicas de overclocking, entretanto, pelo fato de elevarem o hardware ao seu limite, a execução de cargas de trabalho mais intensivas torna-se impraticável, além de registrarem ganhos significativos na temperatura de operação dos

dispositivos.

Estudos mais recentes demonstram a possibilidade da criação de ambientes distribuídos com dispositivos de baixo consumo que usufruam dos benefícios de tecnologias como a virtualização por containers. Segundo (FAYOS-JORDAN et al., 2019), tal estratégia além de melhorar aspectos como escalabilidade do cluster é também perfeitamente praticável, chegando a conclusão de que ao combinar as técnicas de estruturação do cluster com virtualização por contêineres, é possível a obtenção de um melhor aproveitamento dos recursos computacionais dos dispositivos de baixo consumo. Porém, ao passo que se ganha com a escalabilidade do cluster, também cresce a complexidade de orquestrar os recursos do ambiente distribuído, portanto a escolha de uma ferramenta de orquestração adequada passa a ser importante, pois a mesma impactará no desempenho do cluster como um todo. Assim, os resultados obtidos pelos autores demonstram que dentre as ferramentas de orquestração mais populares, Docker Swarm e Kubernetes, a utilização do Docker Swarm com o cluster de Raspberry Pi se demonstrou em torno de 10% mais eficiente no quesito de utilização de recursos dos dispositivos.

O trabalho realizado por (JOHNSTON et al., 2018) traz uma análise comparativa entre a utilização dos computadores de placa única e computadores de arquiteturas tradicionais. Os autores relatam que qualidades como tamanho compacto, baixo consumo de energia e preço tornam esses dispositivos atrativos, o que estimula a realização de trabalhos visando investigar a viabilidade de utilização desse hardware como alternativa ao hardware tradicional. O trabalho também traz um levantamento de algumas das principais aplicações e casos de uso nos quais a utilização dos computadores de placa única podem ser considerados como uma opção, citando cenários como computação de borda, portabilidade de clusters e infraestrutura dos data centers da próxima geração. Como conclusão, os autores relatam que além do já comprovado êxito no processamento da computação de borda e IoT, esse tipo de dispositivo apresenta um paradigma promissor quanto à infraestrutura de ambientes distribuídos.

O trabalho de (STEFFENEL.; CHARãO.; ALVES., 2019) explora a utilização de um cluster de computadores de placa única juntamente da tecnologia de virtualização por contêineres para prover um ambiente distribuído destinado à execução de aplicações científicas. Foi utilizada uma aplicação de computação de alto desempenho, ou HPC (do inglês, High Performance Computing), para a realização da experimentação destinada a comparação do ambiente distribuído composto por um cluster de Raspberry Pi e um cluster de máquinas virtuais. Os resultados obtidos apontaram que o uso do cluster de Raspberry se provou uma boa alternativa quando aliado à tecnologia de conteinerização, já que permitiu associar o baixo custo e fácil manutenabilidade dos computadores de placa única com a flexibilidade de implantação propiciada pelos containers Docker. Ademais, mesmo o desempenho do cluster de Raspberry não sendo páreo para o poder computacional dos tradicionais clusters usados no âmbito de HPC, os resultados obtidos da execução da apli-

cação científica foram os mesmos em ambos ambientes, diferenciando-se apenas pelo tempo de processamento maior, porém aceitável, no cluster de Raspberry.

O trabalho de (BOURHNANE et al., 2020) explora a utilização de um cluster de baixo consumo constituído por placas Raspberry Pi, como uma alternativa que obedece aos conceitos da computação verde na estruturação de um ambiente distribuído para análise e processamento de Big Data com a plataforma Hadoop. Foram executados experimentos comparando o desempenho do cluster de Raspberry Pi com o desempenho de um cluster tradicional durante a execução de benchmarks. Os resultados demonstraram que mesmo o cluster tradicional tendo alcançado resultados melhores, a diferença não inviabiliza a utilização de um cluster de computadores de placa única para essa finalidade, tendo em vista que a eficiência energética dos cluster de Raspberry Pi é consideravelmente maior que a do cluster tradicional. Em uma linha semelhante, o trabalho realizado por (PAPAKYRIAKOU, 2019) utilizou um cluster de dispositivos Raspberry Pi v2 para a implantação da plataforma Hadoop. A análise dos resultados indicam que, assim como em outros trabalhos semelhantes, os principais gargalos de desempenho do cluster de Raspberry Pi estão relacionados com limitações de hardware como largura de banda de conexão e velocidade de E/S dos dispositivos.

No que tange a utilização dos clusters de computadores de placa única para a implantação de bancos de dados distribuídos, (Richardson; Lin; Pecarina, 2017) realizam um estudo sobre a implantação do banco de dados Cassandra. No trabalho são realizados experimentos para a avaliação fatores como escalabilidade, utilização de RAM e desempenho na execução do benchmark YCSB no banco de dados Cassandra e em um cluster composto por dispositivos Raspberry Pi v2. Os resultados obtidos pela experimentação mostram que, dependendo do contexto, dispositivos como Raspberry Pi podem ser vistos como uma boa alternativa ao hardware comumente utilizado nos clusters de bancos de dados.

Em grande parte dos trabalhos mencionados, a análise de desempenho dos bancos de dados distribuídos é feita sob a perspectiva dos clusters e *data centers* tradicionais. Nos poucos exemplos que a fazem em um cluster de SBCs, a análise fica focada na viabilidade de uso dos dispositivos e sob cenários bastante controlados e com pequenos volumes de dados, o que não reflete a realidade de utilização desses bancos de dados em ambientes de produção. Diferentemente, o presente trabalho visa a utilização dos SBCs na estruturação de um ambiente distribuído o mais próximo possível de um ambiente utilizado em cenários reais, além da condução de experimentos que medem o impacto de diferentes fatores de replicação em bancos de dados distribuídos executando no cluster de baixo consumo proposto, dando continuidade a um trabalho preliminar onde foi analisado o desempenho do banco de dados Cassandra para um fator de replicação fixo (SILVA; LIMA, 2021).

4 ANÁLISE COMPARATIVA DE SISTEMAS DE BANCOS DE DADOS DISTRIBUÍDOS EM DISPOSITIVOS DE BAIXO CONSUMO

Os trabalhos relacionados mencionados no Capítulo 3 corroboram que os dispositivos de baixo consumo possuem características interessantes a serem exploradas na composição de infraestruturas para ambientes distribuídos. Em alguns casos, são vistos até mesmo como potenciais alternativas às infraestruturas comumente empregadas nos data centers, visto que mesmo com limitações de poder computacional, suas demais características como o baixo consumo energético, baixo preço e tamanho compacto ainda se demonstram promissoras para estudos nessa linha. Trabalhos e experimentos que exploram a viabilidade do uso de clusters de dispositivos como os SBCs para implantação de ambientes distribuídos, muitas vezes são motivados pelo fato de que os sistemas distribuídos se beneficiam de uma granularidade maior das cargas de trabalho, o que faz com que um cluster, mesmo que composto por nós de processamento com menor capacidade computacional, possa ter seu desempenho global compensado pela quantidade de nós utilizados. Os bancos de dados distribuídos são bons exemplos dessa característica, já que a escalabilidade, granularidade do trabalho e a distribuição tendem a influenciar o desempenho desses sistemas como um todo.

Pensando nisso, no presente trabalho foram conduzidos experimentos com o intuito de analisar o desempenho e comportamento de bancos de dados distribuídos executando sobre um cluster de dispositivos de baixo consumo. Na experimentação conduzida, os bancos de dados foram submetidos a execuções de cargas de trabalho de um benchmark específico para bancos de dados, sendo analisadas métricas como tempo de execução, latência e throughput (em tradução livre, taxa de transferência).

A escolha dos bancos de dados foi baseada na classificação proposta no Teorema de CAP, sendo escolhido um banco de dados de cada grupo: Cassandra representando AP, Hbase representando CP e o PostgreSQL + Citus representando CA. A decisão de utilizar o Teorema de CAP na escolha foi motivada pelo fato de que as 3 possíveis categorias permitem que os bancos de dados sejam agrupados de acordo com seu comportamento em ambientes distribuídos, considerando características como tolerância a falhas, particionamento dos dados, rede, distribuição e disponibilidade dos dados. Além disso, atualmente esse teorema tem sido cada vez mais aplicado na definição da arquitetura e topologia de sistemas modernos e de grande escala, como é o caso por exemplo, das arquiteturas baseadas em microsserviços, que são caracteristicamente distribuídas e altamente escaláveis. Outro fator que motivou a utilização do teorema neste trabalho, é o fato de que as 3 categorias também representam os cenários mais recomendados para utilização de cada banco de dados, pois as peculiaridades e escopos de negócio dos sistemas refletem requisitos que são ou não indispensáveis na escolha das tecnologias a serem utilizadas.

Desta forma, o intuito da experimentação proposta é prover, através da análise comparativa dos desempenhos dos bancos de dados, os fatores positivos, negativos e a viabilidade da utilização dos bancos de dados das 3 categorias do teorema de CAP em um cluster de dispositivos de baixo consumo, sendo os resultados um indicativo de quais cenários que a adoção desse tipo de cluster pode ser uma alternativa atrativa.

4.1 METODOLOGIA

Nesta seção, serão apresentadas as especificações de software e hardware utilizados, bem como a descrição de cada um dos os cenários de teste avaliados neste trabalho.

4.1.1 Especificações de Software

O sistema operacional adotado é o Hypriot OS (versão 1.11), um sistema baseado na distribuição Linux Debian, porém com otimizações para a execução da plataforma Docker em dispositivos com processadores com arquitetura ARM.

Ao optar-se pelo uso de dispositivos não tão comuns para a composição de um ambiente distribuído, é comum o surgimento de alguns obstáculos, como dificuldades para gerenciar um ambiente distribuído formado por dispositivos com características de hardware heterogêneas e a dificuldade de implantação de uma aplicação, pois há bibliotecas e dependências de software distintas para cada arquitetura. Pensando nisso, no intuito de minimizar tais problemas foi utilizado o Docker Swarm (Docker versão 19.03.15) para o gerenciamento e orquestração dos recursos, que abstrai algumas tarefas de gerenciamento dos nós do cluster, bem como facilita a instalação, configuração e dimensionamento dos recursos.

Na execução dos experimentos foram utilizados 3 bancos de dados: Cassandra, Hbase e PostgreSQL (com a extensão Citus).

O Cassandra utilizado é o da versão 3.11.14, o qual foi instalado em uma imagem Docker personalizada [1] baseada na imagem oficial (Docker Community, 2020) disponível no DockerHub. A criação da imagem personalizada foi necessária para possibilitar o suporte da arquitetura ARMv7 que até o momento não possuía suporte oficial. Além disso na nova imagem foram incluídos alguns scripts para melhorar o suporte de recursos de elasticidade do Docker Swarm, recursos como escalar o cluster via CLI docker e sincronização da inicialização de novos nós Cassandra no cluster.

O Hbase utilizado é o da versão 2.2.6, juntamente com o Hadoop versão 3.2.1. Foi

[1] https://github.com/LSC-RPi-Cluster/cassandra-armhf

criada uma imagem Docker personalizada[2] baseada nas imagens disponíveis no projeto (Big Data Europe, 2020), porém foram necessárias alterações significativas para possibilitar o devido suporte à arquitetura ARMv7 e ao Docker Swarm, visto que as imagens utilizadas como base não permitiam nenhuma dessas funcionalidades.

O PostgreSQL foi utilizado em sua versão 13.2 e a extensão Citus na versão 10.1.2. A imagem Docker utilizada foi adaptada dos arquivos da imagem oficial (Citus Data, 2021a), sendo adicionado na imagem personalizada[3] o suporte à arquitetura ARMv7 e ao Docker Swarm, cujas imagens oficiais não permitiam até então. Ainda, foi necessária a criação de uma versão adaptada do serviço *membership-manager* (Citus Data, 2021b) para possibilitar que os recursos de descoberta de nós Citus em um cluster Docker Swarm, além de funcionalidades para escalabilidade do cluster.

Para a realização dos benchmarks foi escolhido o Yahoo! Cloud Serving benchmark (YCSB)(COOPER et al., 2010), pois no que se refere a comparação de bancos de dados distintos, tanto relacionais quanto os não-relacionais (NoSQL), o YCSB é um dos benchmarks mais citados além de ser referência na literatura, sendo que sua ampla adesão justifica-se por ser uma ferramenta cujos testes são padronizados, possui boa usabilidade e suporta uma vasta gama de bancos de dados.

O YCSB inclui um conjunto pré definido de cargas de trabalho, denominados “workloads” e identificados por letras, que descrevem as características de execução, número de registros, número de operações e porcentagem de operações a serem executadas no benchmark. Dentre os seis workloads disponíveis no YCSB, neste trabalho foram utilizados quatro, os quais estão listados a seguir:

- **Workload A:** 50% das operações são leituras e 50% das operações são escritas.
- **Workload B:** 95% das operações são leituras e 5% das operações são escritas.
- **Workload C:** 100% das operações são leituras.
- **Workload D:** Leituras dos registros mais recentemente escritos.

4.1.2 Especificações de Hardware

O cluster de dispositivos de baixo consumo é composto por 15 Raspberry Pi v3 interconectados via rede por um switch de 1Gbps de velocidade. Para o armazenamento dos dados foram utilizados unidades de armazenamento flash de 32 GB de capacidade conectados às interfaces USB de cada Raspberry. A Tabela 4.1 sumariza as configurações de hardware da versão do Raspberry utilizada.

[2]https://github.com/LSC-RPi-Cluster/HBaseSwarmRPI
[3]https://github.com/LSC-RPi-Cluster/postgres-citus-pi

Especificação	Raspberry Pi 3 B
SoC	Broadcom BCM2837
CPU	Cortex-A53 64-bit 1.2GHz quad-core
Arquitetura	ARMv7
RAM	1 GB
Interface Ethernet	300 Mbps

Tabela 4.1 – Especificações de hardware do Raspberry Pi 3 B.

4.2 CENÁRIOS ANALISADOS

Na experimentação foram utilizados 3 clusters de bancos de dados distintos: Cassandra, PostgreSQL com Citus e Hbase. Como cada um dos bancos de dados são arquiteturalmente diferentes e a implantação distribuída de cada um possui suas particularidades, foram utilizados três cenários com topologias distintas para cada cluster.

Em todos os cenários o cluster consiste em 15 dispositivos Raspberry Pi executando um único container cada, representando um nó do cluster de bancos de dados em questão. Para o armazenamento, o diretório no qual a instância de banco de de dados realiza a persistência dentro do container é mapeado para um volume Docker que é montado em um diretório dentro da unidade de armazenando flash no RPi hospedeiro.

Por executarem como uma stack do docker swarm, todos os containers estão interconectados através de uma rede de overlay, permitindo o isolamento entre a rede interna da stack do Swarm e rede externa dos dispositivos hospedeiros. Desta forma, para possibilitar que o YCSB se conecte como cliente aos clusters de bancos de dados, é necessário que os endereços de IP de cada um dos containers estejam visíveis ao benchmark e, para evitar a necessidade de acrescentar mais um serviço de proxy na stack, optou-se por executar o YCSB também de dentro de um container conectado a rede de overlay.

No monitoramento da utilização dos recursos de hardware de cada RPi do cluster foi utilizado o Prometheus, um sistema de monitoramento e alertas que utiliza uma base de dados de séries temporais para o armazenamento dos dados coletados. A coleta das métricas é realizada através dos exporters, que são responsáveis pela captura das métricas e exposição dos dados via API em cada dispositivo para que o Prometheus server possa realizar a coleta. Para hospedar tanto o YCSB quanto o Prometheus, foi alocado um computador exclusivo para tais finalidades, o qual é referenciado nas topologias como servidor de monitoramento (Monitoring Server) e não participa da composição de nós dos clusters de bancos de dados.

Nas topologias dos três cenários o servidor Prometheus executa em um container Docker no servidor de monitoramento e os exporters são executados em cada RPi como um processo em segundo plano diretamente no sistema operacional do hospedeiro. Diferentemente dos demais containers, o servidor Prometheus não é executado como um serviço do Docker Swarm e por isso não é parte da rede de sobreposição, possibilitando

que os Exporters sejam acessíveis pelo Prometheus por meio da rede local que interconecta todos os dispositivos físicos citados nas topologias.

A seguir são ilustradas as topologias de cada um dos cenários analisados no trabalho.

4.2.1 Topologia do Cluster Cassandra

Como o Cassandra é nativamente distribuído e adota uma topologia em formato de anel para a interação entre os nós, a estrutura do cluster não é tão complexa. Em um cluster Cassandra não há um nó com responsabilidades administrativas, portanto, todos nós possuem responsabilidades iguais e todos participam na persistência de dados, entretanto, é necessário que um ou mais nós desempenhe o papel de "seed" no cluster. No momento em que um novo nó é incluso no cluster ele não possui conhecimento de todos os nós participantes da topologia, então esse novo nó, ainda durante a sua inicialização, contacta um nó que já é parte do cluster para "aprender" como os demais nós ativos estão interconectados, esse nó que é contactado é um nó seed. Os nós seed devem ser especificados no arquivo de configuração do Cassandra para que os novos nós saibam com qual nó integrante do cluster se comunicarem para obtenção de informações sobre topologia. Geralmente os nós seed são especificados antes da inicialização do cluster, porém um nó ativo pode ser promovido a seed a qualquer momento. Na topologia do presente trabalho foi fixado um único nó como seed, confome a Ilustração 4.1.

Em virtude das limitações de hardware dos RPis, foi necessária a tomada de algumas decisões para o bom funcionamento do cluster Cassandra. Para contornar as exceções de memória (*out of memory exceptions*), foi necessário definir valores de heap da máquina virtual Java (JVM) de acordo com a quantidade de memória disponível nos dispositivos, assim, o valor da configuração "*maximum heap size*" foi definido para 430MB e o valor de "*size of the new generation*" para 142MB.

Outro problema que frequentemente ocorre quando o Cassandra é executado em clusters com limitações hardware e com vários nós, são as exceções de timeout (em tradução livre, tempo esgotado) de escrita e de leitura, que geralmente ocorrem quando um ou mais nós não conseguem responder a requisição dentro de um tempo limite estabelecido, conforme descrito por (FIGUIèRE, 2014) . Para contornar esse problema, a solução adotada foi definir um tempo de timeout suficientemente grande para que todos os nós do cluster consigam lidar com as requisições. Neste caso, foi estabelecido o valor de 60000 milissegundos para os timeouts de escrita e leitura.

Para o armazenamento, foi mapeado um volume do diretório */var/lib/cassandra* de cada contêiner para o diretório em que a unidade de armazenamento flash estava montada no SO hospedeiro.

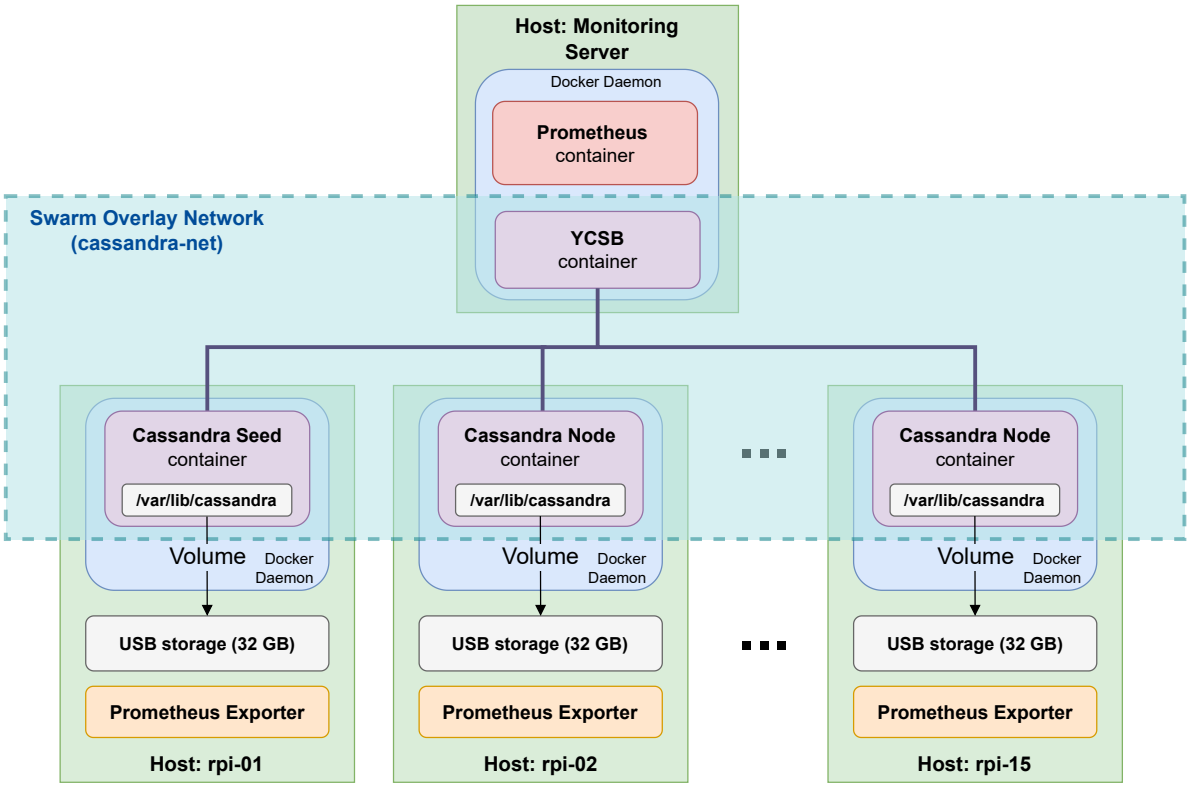


Ilustração 4.1 – Topologia do cluster Cassandra.

Fonte: O autor.

Como não há distinção de responsabilidades entre os nós, as consultas de dados realizadas pelo YCSB não são direcionadas a nenhum nó específico, sendo que durante a execução dos benchmarks, qualquer nó cassandra pode ser o responsável por lidar com a requisição.

4.2.2 Topologia do Cluster PostgreSQL e Citus

Na topologia do PostgreSQL com Citus da Ilustração 4.2, existem dois tipos de nós: o trabalhador Citus (Citus worker) e o Coordenador Citus (Citus Coordinator). No cluster deste cenário de teste, dos 15 RPis, 14 executam containers de trabalhadores Citus e apenas um executa um container de coordenador Citus.

Em um cluster Citus as consultas são feitas apenas para um nó coordenador, pois ele que se encarrega de analisar a consulta e gerar o melhor plano para distribuir o trabalho dentre os nós trabalhadores. Assim, as consultas realizadas pelo YCSB na execução dos benchmarks são todas direcionadas ao nó executando o coordenador Citus.

Como nas topologias dos demais cenários, o diretório da unidade de armazenamento flash foi mapeado para o diretório contendo todos arquivos do banco de dados (o que inclui os próprios dados persistidos). Como todo nó Citus é na verdade uma instância

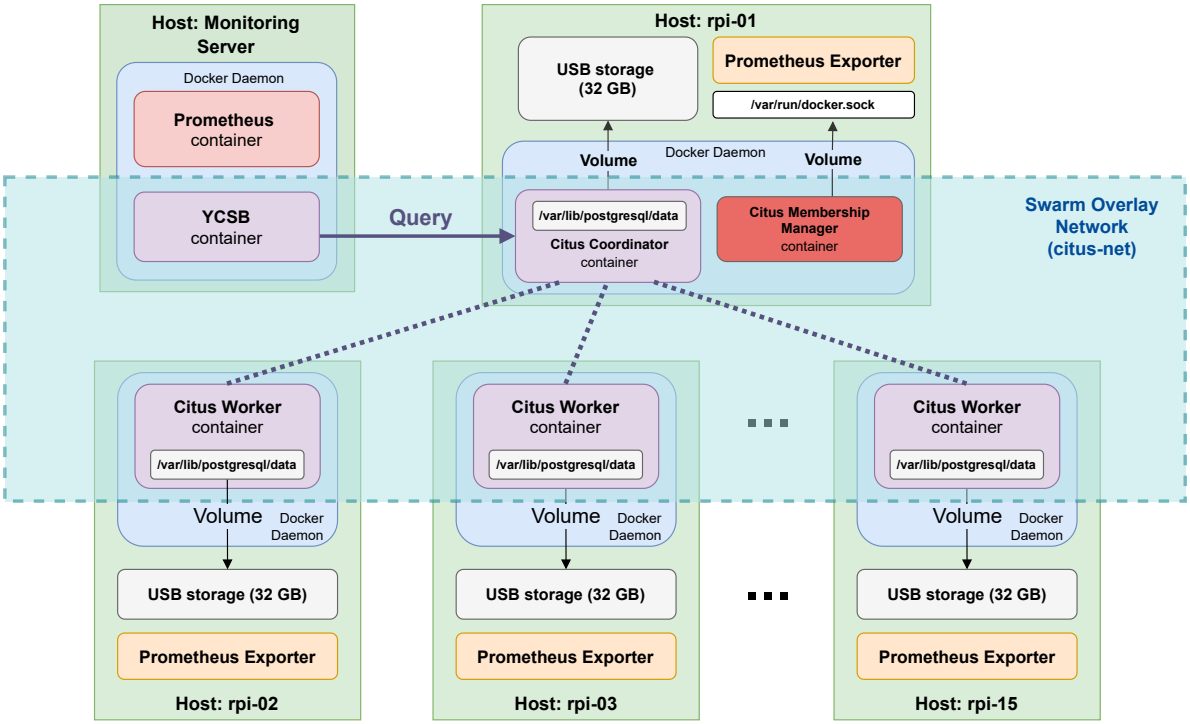


Ilustração 4.2 – Topologia do cluster PostgreSQL com Citus.

Fonte: O autor.

do postgres, o diretório mapeado pelo volume é */var/postgresql/data*.

Uma particularidade que pode ser percebida na ilustração da topologia, é a existência de um container a mais sendo executado no mesmo RPi que executa o coordenador Citus, tal contêiner refere-se ao serviço *membership-manager*. O propósito do *membership-manager* é monitorar constantemente a rede de sobreposição do Docker Swarm para identificar o ingresso e saída de nós no cluster, isso porque pelo fato do PostgreSQL por padrão não ser um banco de dados distribuído ele não possui recursos nativos para escalar horizontalmente um cluster. Assim, o *membership-manager* foi uma forma que os desenvolvedores do Citus encontraram para possibilitar fácil dimensionamento dos trabalhadores Citus, sem a necessidade de manualmente especificar que um nó está entrando ou saindo do cluster. O *membership-manager* utilizado neste trabalho é uma modificação do criado pela equipe desenvolvedora do Citus, as alterações realizadas são para permitir suporte ao Docker Swarm. Todo funcionamento desse serviço é baseado em verificar constantemente o status dos containers em execução na stack Swarm por meio de consultas à API do Docker, acessada via Socket Unix e que é acessível de dentro do container através de um volume mapeado para o diretório */var/run/docker.sock* no RPi hospedeiro.

4.2.3 Topologia do Cluster Hbase

O cluster do cenário de teste com Hbase foi estruturado para a execução do modo Fully-ditributed do banco de dados, sendo necessária a configuração do ambiente para execução dos serviços complementares para a obtenção do Hbase em sua versão de fato distribuída. Como mencionado anteriormente na Seção 2.4, no modo Fully-ditributed além das instâncias do Hbase (Hmaster e HRegionServers), faz-se necessário a execução também do Zookeeper e do HDFS para suportar a comunicação do nós fisicamente distribuídos e a replicação dos Hfiles. Desta forma, na topologia utilizada neste trabalho, optou-se por executar todos serviços necessários em um único contêiner Docker, no intuito de obter um melhor proveito dos recursos de gerenciamento de memória e processos da JVM, visto que o Hbase e o HDFS são aplicações Java e executá-los em contêineres distintos acarretaria em uma sobrecarga desnecessária, tanto de gerenciamento de mais um contêiner quanto da execução de uma JVM para cada processo.

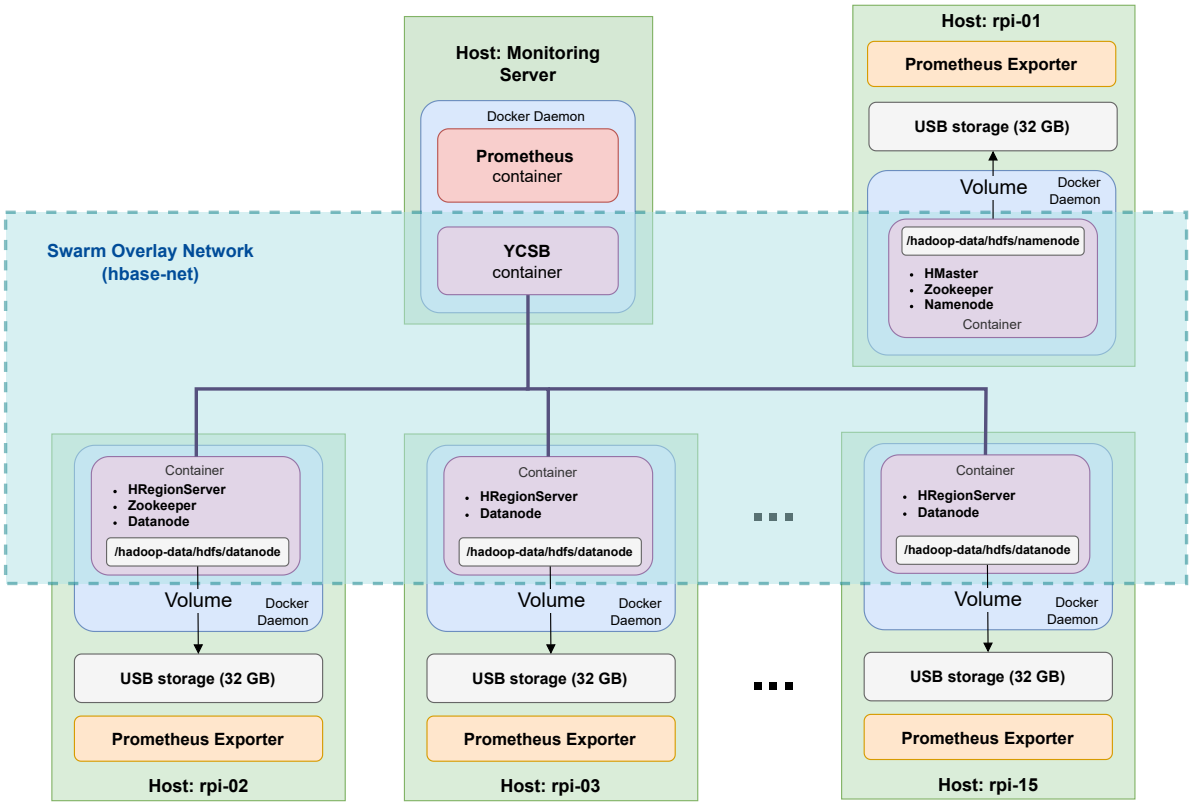


Ilustração 4.3 – Topologia do cluster Hbase.

Fonte: O autor.

Conforme pode ser observado na topologia da Ilustração 4.3, existe apenas um nó executando o HMaster juntamente do Namenode e este não faz parte do armazenamento de dados distribuído, pois tanto o Namenode quanto o Hmaster possuem responsabilidades apenas de gerenciamento do cluster. Os demais nós da topologia participam ativamente da persistência de dados e cada um executa um contêiner com uma instância

HRegionServer e um Datanode. Para o controle de estado do cluster optou-se pela execução do Zookeeper nativo do próprio Hbase, o HQuorumPeer. Para fins de prover um nível razoável de tolerância a falhas, 3 dos nós executam um processo HQuorumPeer além dos demais processos, sendo esses o nó do Hmaster e dois outros nós HRegionServers.

Durante a execução do YCSB, apenas os nós executando um HRegionServer participam das operações relacionadas ao tratamento e execução das consultas, sendo que qualquer HRegionServer do cluster poderá ser responsável por lidar com a operação em questão. Os HFiles com os dados são armazenados no diretório de dados do Datanode de cada container (*/hadoop-data/hdfs/datanode*), o qual é mapeado como um volume Docker para o dispositivo de armazenamento montado no RPi hospedeiro.

4.2.4 Configuração das execuções

A execução de cada workload do YCSB é composta por duas etapas: o carregamento (load) dos dados e a execução do workload. Na etapa de load, o banco de dados a ser testado é populado com um conjunto de registros com valores aleatórios para prover o cenário no qual as consultas dos worloads serão realizadas, sendo o número de registros estabelecido pela configuração “recordcount” do YCSB. Cada registro inserido durante a etapa de load possui 1KB de dados, sendo 100 bytes para cada uma das 10 colunas da tabela de dados usada pelo YCSB. O volume de dados estabelecido para todas as execuções deste trabalho foi fixado em 1GB, ou 1 milhão de registros.

Após a fase de load concluída, a segunda etapa é a execução do workload. Nesta etapa, é onde ocorre de fato a execução do workload sobre o conjunto de dados criado no load e utilizando os parâmetros (porcentagem de leituras e escritas) estabelecidos para cada um. A etapa de load é pré-requisito para a execução dos workloads, entretanto, dependendo dos tipos de operações realizadas, é possível reaproveitar a etapa de load para a execução de mais de um workload, como é o caso dos workloads A, B, C e D que, se executados nessa ordem, podem todos utilizar a etapa de load do workload A, visto que as operações de escrita realizadas por A e B são de consultas de update ao banco de dados e por isso não afetam a número de registros na base dados, sendo somente D o workload que realiza novas consultas de inserção de dados novos.

Como mencionado anteriormente, neste trabalho foram utilizados os workloads A, B, C e D. Foram realizadas 30 execuções de cada workload, sendo cada um configurado para realizar 10000 operações no banco de dados. Nos resultados foi considerada a análise de variância (ANOVA) sobre os valores de tempo de execução, throughput e latência de cada banco de dados, para o devido cálculo da significância estatística das diferenças aplicando um Teste F. A análise considera a média das 30 execuções de cada configuração e um intervalo de confiança de 95% (linhas verticais sobre as médias dos gráficos

de barras). Para a comparação das diferenças foi utilizado o teste Tukey HSD (do inglês, Honestly Significant Difference) e ponto de corte de significância P < 0,05 (WITTE; WITTE, 2017).

Além das configurações de execução dos workloads, para cada banco de dados o fator de replicação foi variado entre os valores 1, 2 e 3 para cada conjuto de configurações de execução do YCSB. A permutação de todas configurações executadas é dada por: $(4\ workloads * 30\ execucoes * 3\ fatores\_de\_replicacao * 3\ bancos\_de\_dados)$, totalizando 1080 execuções.

4.3 RESULTADOS

Nesta seção são apresentados os resultados experimentais obtidos da execução do benchmark YCSB em cada um dos cenários descritos e para cada configuração de execução.

4.3.1 Carga dos dados para execução do benchmark

A etapa de load do YCSB foi analisada relacionando cada um dos 3 bancos de dados e os diferentes valores de fator de replicação. O gráfico 4.1 demonstra essa relação para os tempos de execução da etapa de load para cada configuração.

Analisando o gráfico 4.1, é possível notar que o Cassandra não sofre impacto com o aumento do número de réplicas dos dados, apresentando tempos de execução semelhantes para os 3 valores de fator de replicação. Em contrapartida, o Citus é fortemente impactado com o aumento do fator de replicação.

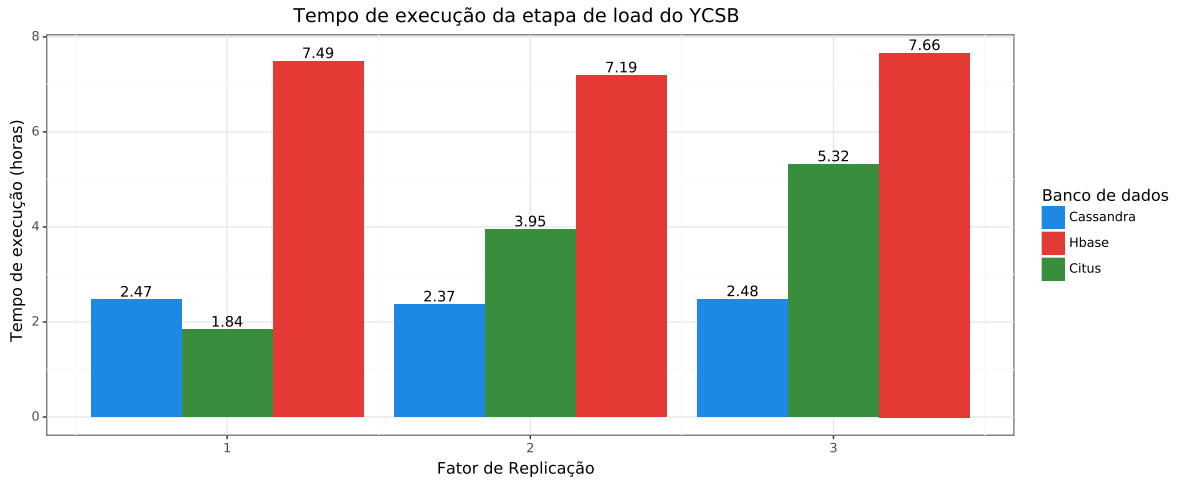


Gráfico 4.1 – Tempo de execução das etapas de load (em horas) para cada fator de replicação

Fonte: O autor.

O Hbase, por sua vez, não demonstra ser influenciado de forma direta pelo número de réplicas, porém demonstra os maiores tempos de execução em todos cenários. Em análise geral dos tempos de execução da etapa de load, o Cassandra obteve os melhores resultados com baixos valores em todos os cenários em comparação com os demais bancos de dados e o Hbase obteve o maior tempo em todos os cenários. Citus apresenta o melhor resultado apenas para o cenário de fator de replicação 1.

4.3.2 Execução dos Workloads

A partir da execuções dos workloads do YCSB é possível coletar algumas métricas sobre o desempenho do banco de dados nas operações especificadas para o workload em questão, a seguir são apresentadas e comentadas algumas destas métricas.

4.3.2.1 Tempo de execução

Avaliando o tempo médio de execução das execuções dos workloads A, B, C e D do YCSB para cada banco de dados e fator de replicação (Gráfico 4.2) evidencia-se que o Cassandra obteve os menores tempos de execução para os workloads A e B com fatores de replicação 1 e 2, e workload D para os três valores de réplicas.

O Hbase possui resultados melhores em comparação com o Cassandra para o workload C e fator de replicação 3 e resultados estatisticamente equivalentes ao Cassandra no workload C para número de replicas igual a 1 e 2, e no workload A com fator de replicação

igual a 1. No workload B com fator de replicação 3 o Hbase teve resultados ligeiramente maiores que o Cassandra.

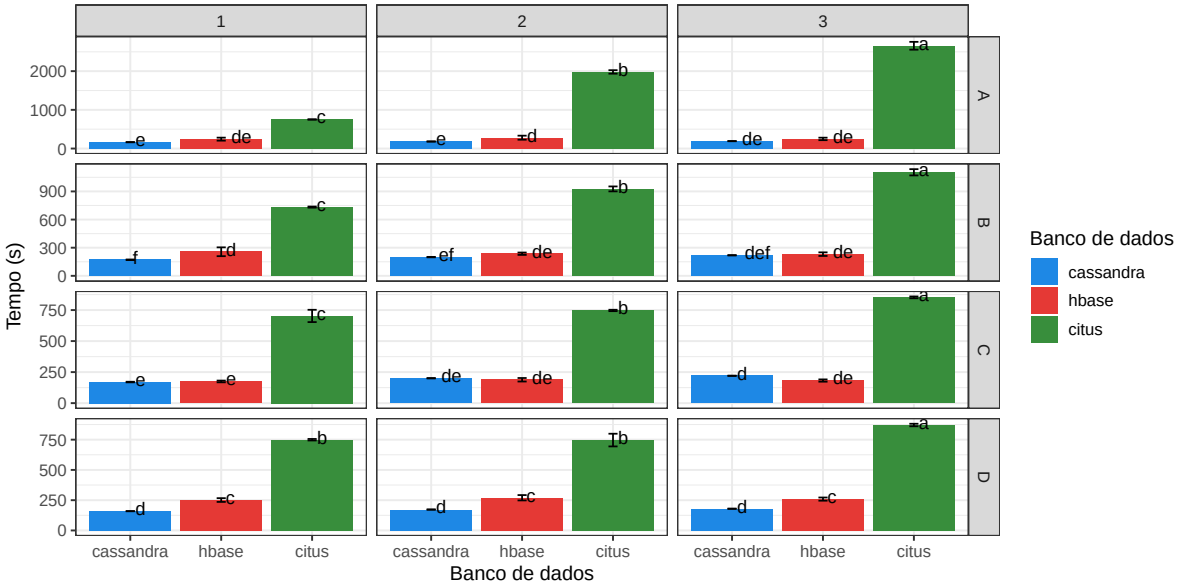


Gráfico 4.2 – Tempos médios de execução de cada workload, banco de dados e fator de replicação. Para tempo total de execução menores valores são melhores. As linhas verticais nas barras representam o intervalo de confiança e as letras indicam o grau de significância estatística das diferenças dos dados de cada seção do gráfico (configuração de execução).

Fonte: O autor.

Citus apresenta os maiores tempos de execução para todos os casos. Com exceção ao workload D e fator de replicação igual a 1, diferentes workloads não afetam o tempo de execução no Citus, porém os tempos são fortemente afetados de forma negativa com o aumento do fator de replicação. Em contrapartida, tanto Hbase quanto Cassandra não demonstram ser impactados pelo aumento do número de réplicas.

4.3.2.2 Throughput

Analisando os resultados de throughput (operações por segundo) dos 4 workloads e os 3 fatores de replicação (Gráfico 4.3) observa-se que de forma geral Cassandra possui os melhores resultados de throughput e Citus possui os valores menores em todos cenários.

O maior número de outliers e maior variabilidade nos resultados de throughput são observados nos resultados do Hbase, sendo os outliers uma evidência da influência da distribuição dos dados nas operações executadas no Hbase.

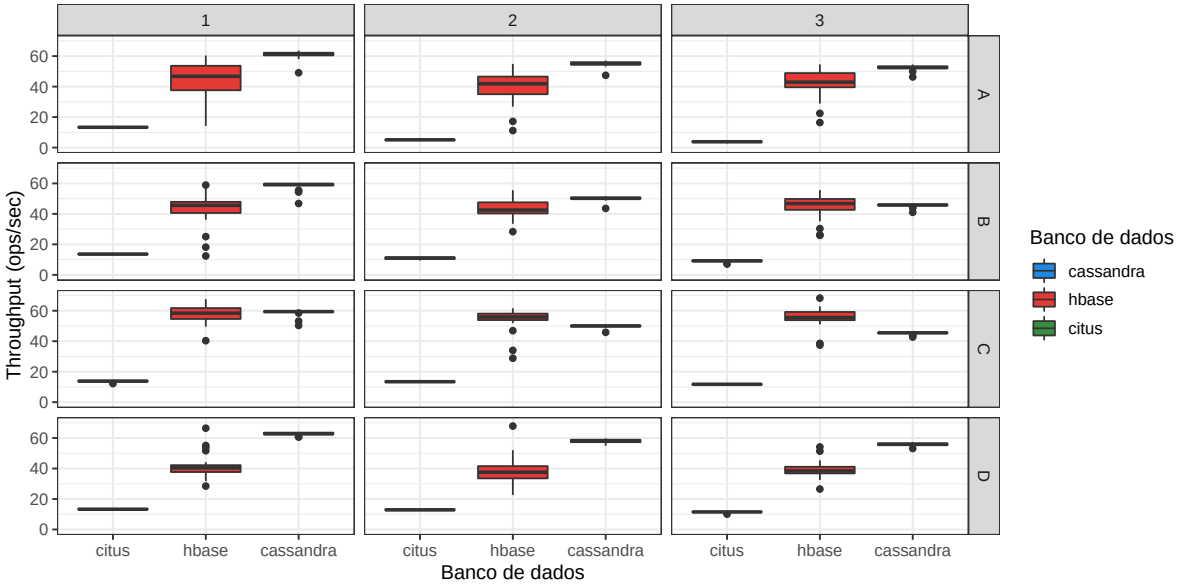


Gráfico 4.3 – Throughput médio de execução de cada configuração de fator de replicação, workload e banco de dados. Para resultados de throughput valores maiores são melhores. A linha central nas caixas representa a mediana dos valores, a amplitude do gráfico representa a dispersão dos dados e os pontos representam os outliers.

Fonte: O autor.

Citus apesar de apresentar os piores resultados de throughput em todos os casos, possui constância em todas execuções, com baixa variabilidade e com raros outliers.

No gráfico 4.4 as médias de throughput são apresentadas sob o ponto de vista da análise de variância dos dados e seu grau de significância estatística dentre os resultados. Dele é possível concluir:

- Cassandra obteve melhores valores de throughput que o Hbase no workloadas A e D para todos fatores de replicação.
- Hbase obteve melhores valores de throughput que o Cassandra no workload C com fatores de replicação 2 e 3 e valor estatisticamente equivalente ao Cassandra no workload B com 3 réplicas.
- Fatores de replicação maiores que 1 afetam negativamente os valores de throughput do Cassandra.
- Hbase não demonstra sofrer significante impacto com o aumento do fator de replicação, apresentando apenas uma letra (c) nos workloads B e D e duas letras (b e c) nos workloads A e C.
- Citus obteve os piores valores de throughput em todos casos, não sendo significantemente afetado pelo fator de replicação, com exceção do Workload A com fator de replicação 1.

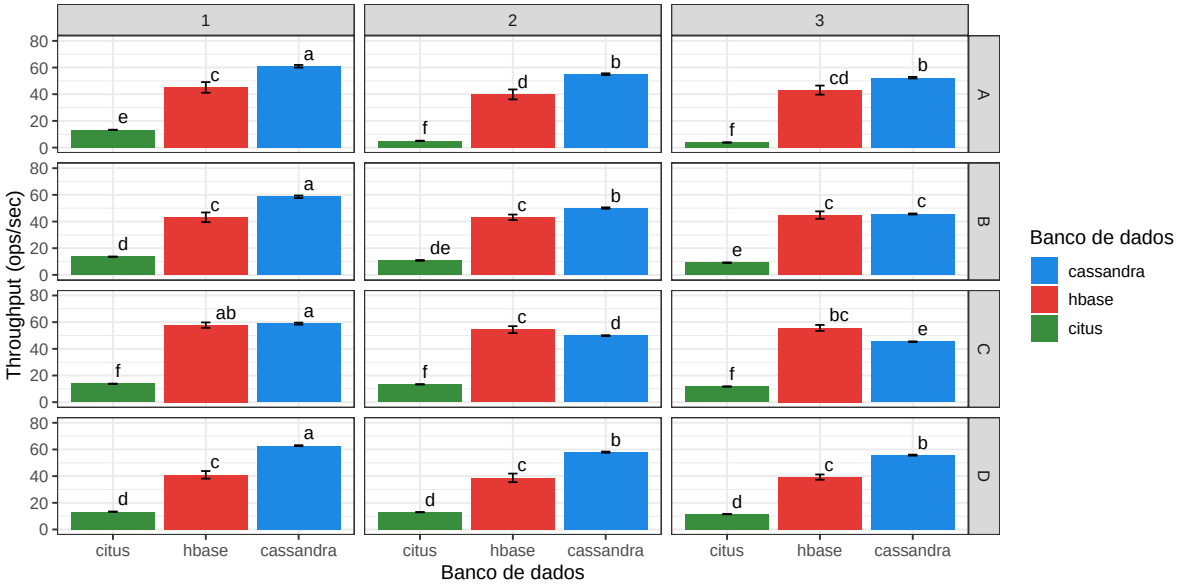


Gráfico 4.4 – Throughputs médios da execução de cada configuração de workload, banco de dados e fator de replicação. Para throughput valores maiores são melhores. As linhas verticais nas barras representam o intervalo de confiança e as letras indicam o grau de significância estatística das diferenças dos dados de cada seção do gráfico (configuração de execução).

Fonte: O autor.

4.3.2.3 Latência

Os resultados de latência das operações de leitura das execuções dos workloads (gráfico 4.5) mostram que, de modo geral, Cassandra e Hbase não demonstram diferenças consideráveis entre seus resultados, com exceção do Workload D no qual o Hbase apresentou valores ligeiramente maiores para os três fatores de replicação. Citus, por sua vez, apresentou os maiores valores de latência de leitura em todos cenários, sendo estes fortemente influenciados negativamente com o aumento do fator de replicação.

A variação do workloads não se apresenta como um fator que impacta na latência de leitura nos bancos de dados e diferentes fatores de replicação impactam apenas nos valores do Citus.

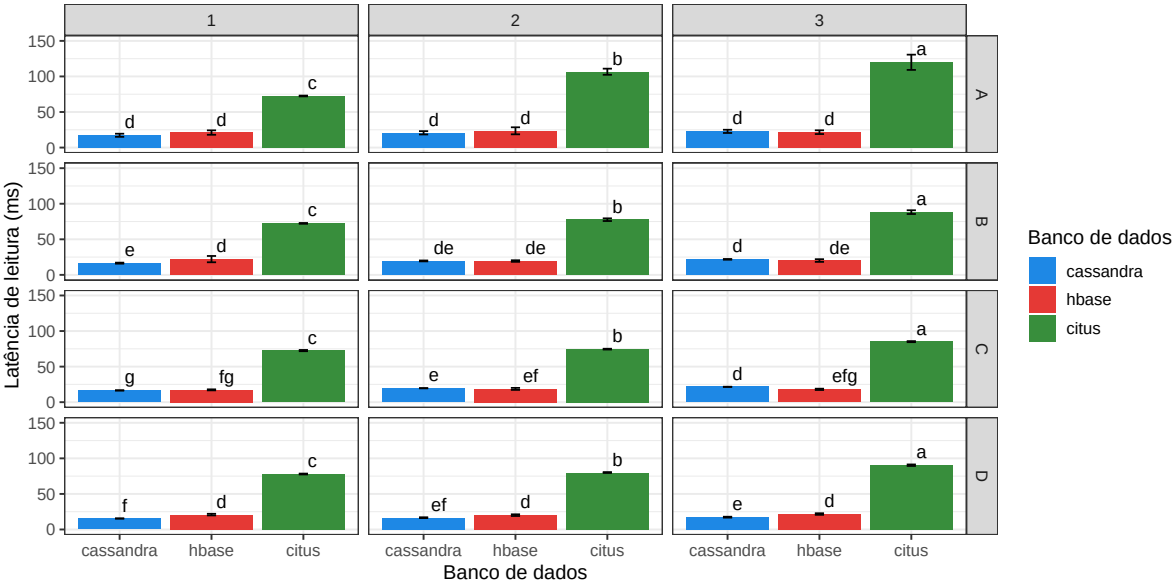


Gráfico 4.5 – Latências médias para operações de leitura da execução de cada configuração de workload, banco de dados e fator de replicação. Para latência valores menores são melhores. As linhas verticais nas barras representam o intervalo de confiança e as letras indicam o grau de significância estatística das diferenças dos dados de cada seção do gráfico (configuração de execução).

Fonte: O autor.

Sob a perspectiva dos resultados de latência de operações de escrita (gráfico 4.6) o Cassandra obteve os melhores resultados (mais baixos) em todos os casos, além de não demonstrar influência de diferentes fatores de replicação. Em contrapartida, o resultados do Citus para os workloads A e B (consultas de update) sofrem significativa influência do aumento do número de réplicas. O Hbase não é afetado pelo fator de replicação nos workloads A e B, mas é afetado no workload D com duas réplicas. Os resultados do Citus para o workload D são inconclusivos devido a erros em alguns casos, desencadeados pela leitura de um dado recentemente escrito por uma operação em batch.

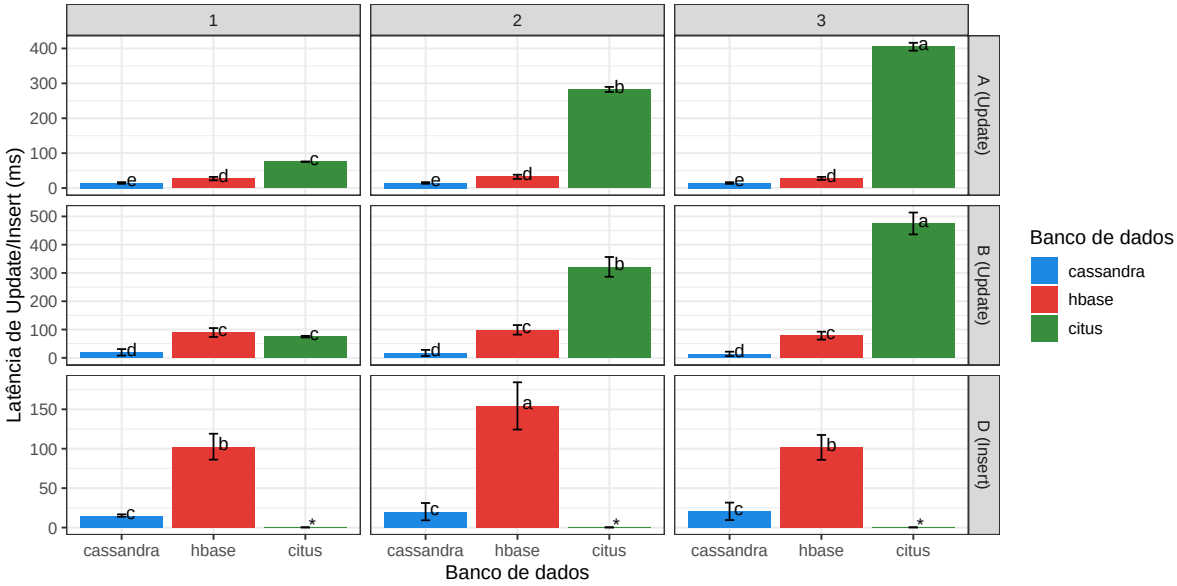


Gráfico 4.6 – Latências médias para operações de escrita (update/insert) da execução de cada configuração de workload, banco de dados e fator de replicação. Para latência valores menores são melhores. As linhas verticais nas barras representam o intervalo de confiança e as letras indicam o grau de significância estatística das diferenças dos dados de cada seção do gráfico (configuração de execução).

Fonte: O autor.

4.3.3 Utilização de recursos

Todas topologias de cluster utilizadas nas execuções dos workloads do YCSB contaram com a presença de um exporter do Prometheus, responsável pela coleta de métricas de uso de recursos. Sendo assim, é possível sumarizarmos resultados de uso de alguns recursos considerados relevantes durante as execuções, neste caso, foram elencados o uso de RAM, CPU e armazenamento (distribuição dos dados). Para representar de forma mais fiel os dados de RAM e CPU, foi adotada a mediana dos valores ao invés da média.

4.3.3.1 Distribuição dos dados

Após a etapa de load do YCSB concluída, é possível observar como os dados são distribuídos através dos nós do cluster em cada banco de dados. O gráfico 4.7 permite a análise da homogeneidade dessa distribuição ao relacionar: o espaço ocupado em disco em cada nó do cluster, com o banco de dados e fator de replicação utilizado na etapa de load.

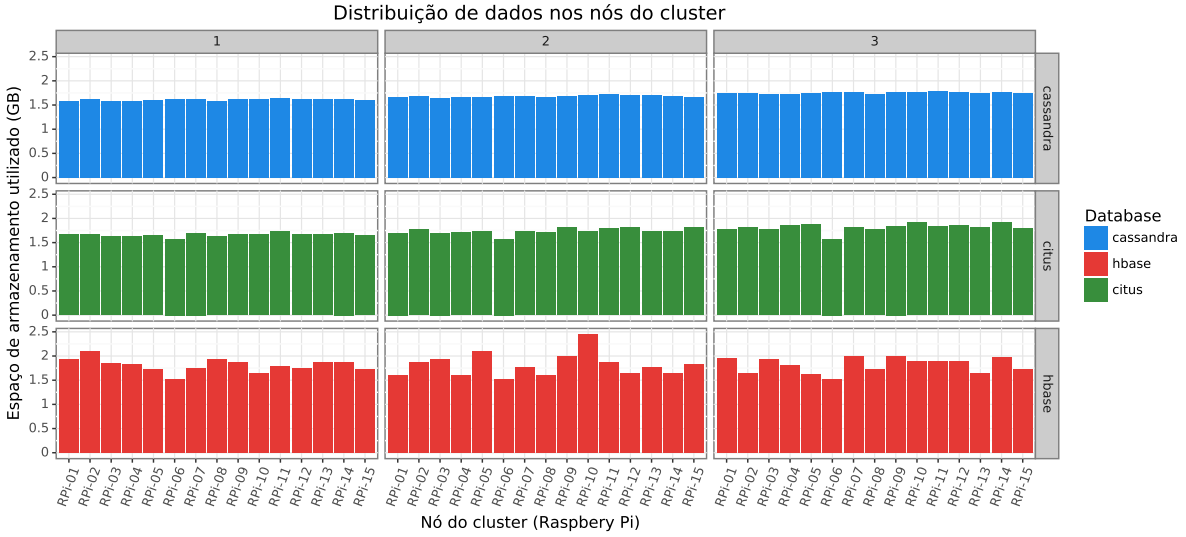


Gráfico 4.7 – Distribuição dos dados entre os nós do cluster. Os segmentos verticais do gráfico representam os diferentes fatores de replicação (1, 2 e 3) e os segmentos horizontais cada um dos bancos de dados. Quanto menor a diferença de altura das barras em cada seção, melhor distribuídos estão os dados.

Fonte: O autor.

O Cassandra apresenta a melhor distribuição dentre os 3 bancos de dados, não havendo influência do fator de replicação na homogeneidade da distribuição através dos nós.

O Citus apresenta distribuição razoavelmente balanceada, não possuindo impacto relacionado ao fator de replicação além do aumento do volume dados. No mecanismo de distribuição do Citus não é executado um rebalanceamento a cada inserção e como é utilizada uma estratégia round-robin para a alocação dos shards, é esperado que a distribuição dos dados não seja ideal, porém mesmo assim resulta em uma distribuição aceitável.

A distribuição do volume de dados no Hbase é a mais heterogênea em comparação com os demais bancos de dados. Isso porque a distribuição é realizada a nível de Regions não a nível de Hfiles, logo é possível que os RegionServers possuam Regions em número e tamanho distintos dentro do cluster. Além do mais, no mecanismo de distribuição do Hbase, um Region só é realocado para um novo nó com menos carga quando esse Region atingir um tamanho máximo estipulado, fazendo com que seja possível a existência de RegionServers com Regions ainda não realocados para outro RegionServer, pois ainda não satisfazem o critério de tamanho máximo para serem redistribuídos.

No que tange o impacto dos diferentes fatores de replicação na distribuição de dados do Hbase, há influência. Observando o gráfico é possível perceber que o volume de dados está menos distribuído com o fator de replicação igual a 2. Essa influência é justificada pelo fato de que ao modificar o volume de dados, consequentemente se modifica o número de Regions a serem distribuídos, assim, nos casos em que o particionamento do volume de dados em diversos Regions resulta em um número de Regions não múltiplo

do número de RegionServers, será inevitável que um ou mais RegionServers possuam um maior número de Regions, desbalanceando a distribuição dos dados no cluster.

Tanto no Citus quanto no Hbase, independente do fator de replicação o nó RPi-06 possui o mesmo volume de dados, pois este é nó adotado como coordinator (Citus) e HMaster (Hbase) no cluster, sendo assim, os dados que são armazenados nesse nó são metadados e informações utilizadas para o gerenciamento do cluster, não fazendo parte do conjunto de dados utilizados pelo YCSB. Somente no Cassandra todos nós do cluster participam do armazenamento.

4.3.3.2 RAM

Os resultados da mediana das métricas coletadas de uso de RAM para todos os nós do cluster durante as 30 execuções de cada configuração, são apresentadas no gráfico 4.8.

De forma geral, Cassandra se destaca com o maior consumo em MB em todos os cenários, sendo os valores próximos à quantidade máxima de RAM disponível para execução de processos nos RPIs ($\approx 750MB$). Maior fator de replicação afeta de forma sucinta o uso de RAM no Cassandra, já o Hbase e o Citus não sofrem influência do número de réplicas. Citus apresenta os valores de consumo mais moderados de RAM, sendo os menor uso e mais estável em todos os casos. Cassandra e Hbase apresentam consumo ligeiramente menor para o Workload D.

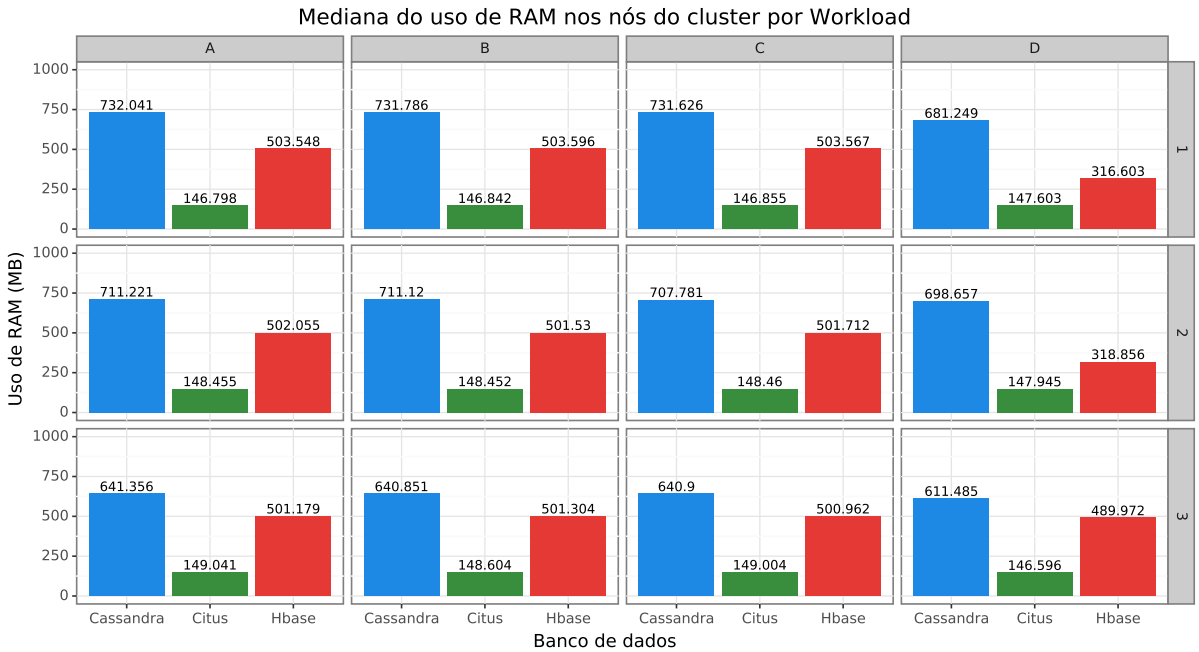


Gráfico 4.8 – Mediana do uso de RAM dos nós do cluster durante as execuções

Fonte: O autor.

4.3.3.3 CPU

No gráfico 4.9 são apresentadas as medianas das métricas coletadas de uso de CPU para todos os nós do cluster durante as 30 execuções de cada configuração.

Os resultados de porcentagem de uso do de CPU são baixos em todos os cenários de execução. Com exceção dos valores do Cassandra para o workload D com fatores de replicação 1 e 2, que resultaram nas porcentagens de 12,09% e 13,61% respectivamente, todas medianas são inferiores a 7%.

Apesar dos valores baixos em mediana, analisando o perfil de uso de CPU individual de cada nó do cluster em uma das execuções (Apêndice A), é possível notar que alguns poucos nós do cluster Citus demandam significativo maior uso de CPU. Hbase há nós com uso de CPU maiores que outros mas não muito destoantes dos demais. No Cassandra as porcentagens de uso de CPU são mais equilibradas entre os nós do cluster.

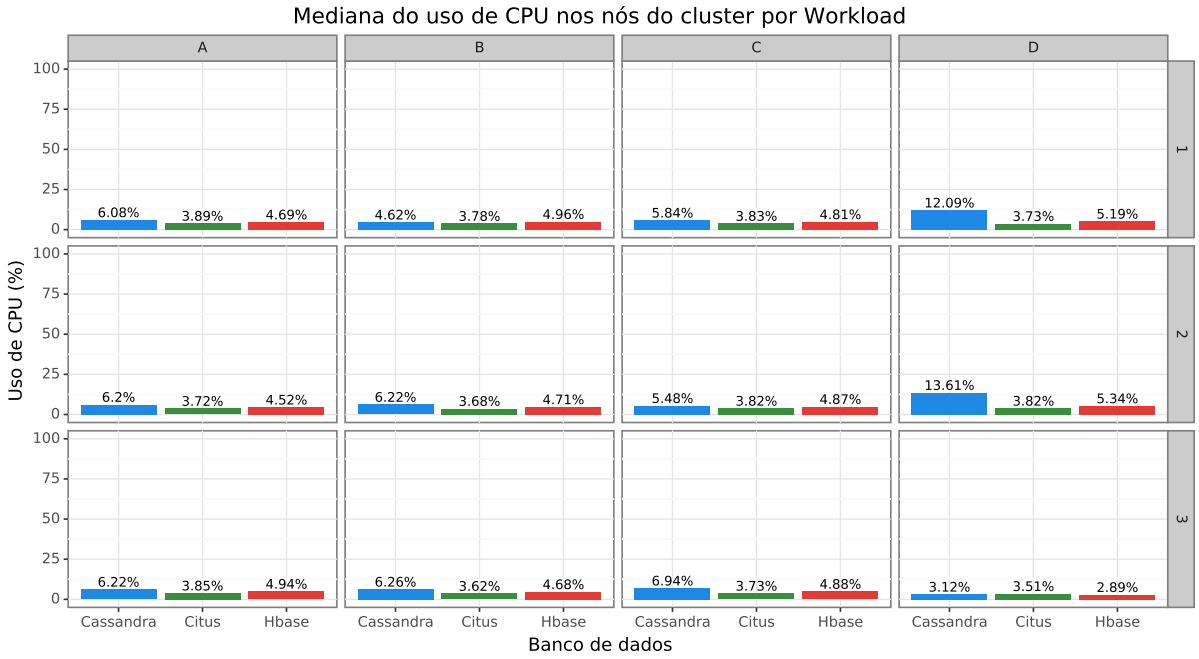


Gráfico 4.9 – Mediana da porcentagem de uso de CPU dos nós do cluster durante as execuções

Fonte: O autor.

4.4 DISCUSSÃO

Os resultados experimentais mostram que na carga de um volume de dados significativo, caracterizado por múltiplas operações de escrita sequenciais, o Cassandra obtém melhores resultados que os demais bancos de dados independente do aumento do nú-

mero de réplicas. O Hbase obteve os piores resultados para esse tipo de tarefa, porém, não demonstra impacto com o aumento do fator de replicação. Tais resultados se dão pelo fato que o Hbase em modo Fully-ditributed executa sobre o Hadoop, portanto as operações de escrita tendem a demorar mais, visto que envolvem também o processo de escrita dos Hfiles em um sistema de arquivos distribuído.

Citus demonstra grande impacto do aumento do número de réplicas em tarefas como carga de dados, isso porque o mecanismo de replicação do Citus consiste basicamente em replicar as operações de escrita para um número de shards igual ao fator de replicação menos 1 (dado original + replicas), logo, durante uma carga de trabalho como a fase de load do YCSB, cada operação de inserção será executada tantas vezes quanto o valor estabelecido para o fator de replicação. Da mesma forma, justifica-se o fato do Citus ter obtido o melhor tempo de execução na etapa de load com uma única réplica, já que a característica de escritas sequenciais dessa tarefa permite que o Citus ao invés de realizar um INSERT para cada registro, reúna vários registros em grupos (os batches) e os insira de uma só vez no banco de dados por meio de uma única operação de COPY do PostgreSQL, dando vantagem ao Citus nos casos que as operações não são multiplicadas pelo fator de replicação. Entretanto, apesar das sobrecargas de operações necessárias para atender as propriedades de um banco de dados relacional, por meio dos recursos de replicação lógica dos shards e consultas distribuídas, Citus consegue fornecer ao PostgreSQL os requisitos de um banco de dados distribuído do tipo CA da classificação de CAP.

No âmbito da execução dos workloads do YCSB, os resultados demonstram que o Cassandra atingiu melhores resultados em workloads orientados a operações de escrita (update e insert), enquanto que o Hbase se destaca em workloads caracterizados por maior número de operações de leitura, como no workload C. Ao que tudo indica, os melhores resultados do Cassandra são decorrentes principalmente por sua arquitetura contar com estruturas de armazenamento integradas, além da garantia de consistência eventual, que permite aos bancos de dados da classificação AP do teorema de CAP sacrificarem temporariamente a consistência das escritas em prol do aumento da vazão e redução de latência das operações. Em contrapartida, bancos de dados da classificação CP, como o Hbase, não abrem mão da consistência e por consequência da sobrecarga que essa garantia demanda. No caso do Hbase, toda escrita ou atualização de um dado requer um novo registro no WAL e, como esse arquivo de WAL está armazenado no HDFS, tal registro é submetido a todas operações de replicação para satisfazer os critérios do sistema de arquivos distribuído.

Os resultados de throughput deixam claro o impacto que as limitações de hardware do Raspberry Pi 3 têm sobre as execuções. Dentre as observações sobre uso de recursos coletadas com o Prometheus, RAM destaca-se como recurso com mais influência nas cargas de trabalho e com maior potencial para ser fonte de gargalos no sistema, visto que em alguns casos, conforme observado para o Cassandra no gráfico 4.8, o uso de RAM em to-

dos cenários está saturado e, nos demais bancos de dados, possui uso significativo. CPU, apesar de apresentar utilização maior em alguns nós do cluster durante as execuções, não se apresenta como um recurso saturado para esse tipo de aplicação.

Em comparação com alguns trabalhos correlacionados, os autores (Tang; Fan, 2016; ABRAMOVA; BERNARDINO, 2013; HENDAWI et al., 2019) relatam resultados de throughput expressivamente maiores, porém com volumes de dados menores e em clusters com poder computacional superior. Por outro lado, os resultados obtidos para latência são comparáveis aos relatados em trabalhos executando os bancos de dados distribuídos em ambientes compostos por máquinas virtuais ou em clusters de alto desempenho (DATASTAX, 2015; KUHLENKAMP; KLEMS; RöSS, 2014; HUANG et al., 2015).

5 CONCLUSÃO

Neste trabalho foram avaliados três bancos de dados executando sobre um cluster composto por 15 dispositivos Raspberry Pi 3 B. A escolha dos bancos de dados se baseou na classificação proposta pelo teorema de CAP, sendo o Cassandra representando a categoria AP, Hbase a categoria CP e PostgreSQL com Citus a categoria CA. A experimentação realizada teve foco em analisar os comportamentos dos bancos de dados durante a execução de cargas de trabalho da ferramenta de benchmark YCSB sobre o ambiente distribuído proposto e para diferentes fatores de replicação. O cluster foi estruturado utilizando-se uma camada de virtualização por contêineres Docker juntamente ao Docker Swarm, que desempenhou o papel de orquestrador, ferramenta de implantação dos serviços e balanceador de carga entre os nós do cluster.

Os resultados obtidos evidenciam que clusters de baixo consumo compostos por SBCs, podem sim atender as exigências de sistemas distribuídos utilizados em cenários reais. O cluster utilizado neste trabalho demonstrou-se capaz de suportar os 15 nós executando significativas cargas de trabalho, além de permitir balanceamento de carga entre os dispositivos. Além do mais, com as 3 topologias propostas foi possível alcançar alta elasticidade por meio dos recursos oferecidos pelo Docker Swarm, permitindo o ingresso ou remoção de nós no cluster em tempo de execução. Também foi possível alcançar alta disponibilidade nas topologias, característica essa provida pelos bancos de dados distribuídos utilizados.

Na execução das cargas de trabalho do YCSB, de modo geral, o banco de dados Cassandra obteve os melhores resultados, destacando-se em workloads orientados a operações de escrita e não demonstrando ser afetado pelo aumento do fator de replicação. Os bons resultados do Cassandra podem ser justificados pelo seu relaxamento na garantia da consistência dos dados, pois adota a estratégia de consistência eventual que lhe poupa da sobrecarga do processo de garantia de uma consistência mais rígida, em prol de uma maior vazão e menor latência nas operações. Em contrapartida, Hbase e Citus não abrem mão de sua forte garantia de consistência e por isso acabam tendo seu desempenho penalizado com o aumento do do fator de replicação, já que além da consistência do dado faz-se necessária a garantia de consistência das réplicas. Contudo, os resultados obtidos são comparáveis aos relatados em alguns trabalhos relacionados em que os bancos de dados distribuídos foram executados sobre ambientes compostos por máquinas virtuais ou em clusters de alto desempenho.

Dos resultados experimentais ainda é possível concluir que existe uma nítida relação entre fator de replicação e o desempenho das operações nos bancos de dados, sendo que apesar do fato que o aumento do fator de replicação acarreta em maior disponibilidade e tolerância a particionamento, consequentemente também penaliza o desempenho. Dito

isso, a escolha de um banco de dados distribuído deve levar em conta não somente o desempenho de operações, mas também o contexto de aplicação e regras de negócio que o mesmo será implantado, visto que cenários nos quais a consistência é fator primordial podem não ser uma boa aplicação para os bancos de dados AP, assim como bancos de dados CP podem não ser ideais para cenários que demandam alta disponibilidade.

No contexto do provisionamento de sistemas distribuídos utilizando dispositivos de baixo consumo, as características dos bancos de dados AP tornam-se essenciais, visto que os SBCs possuem tipicamente um baixo MTBF (Tempo Médio Entre Falhas, na sigla em inglês), estando o cluster suscetível a uma maior probabilidade de ocorrerem as partições descritas pelo teorema de CAP.

Mesmo com os objetivos deste estudo atingidos, ainda existem aspectos que podem ser explorados sobre o uso de bancos de dados distribuídos no cluster de SBCs, o que abre espaço para o desenvolvimento de trabalhos futuros sustentados pelas conclusões neste obtidas. Uma análise sob a perspectiva da escalabilidade do cluster é um dos possíveis temas a ser explorado, visto que, apesar de ter sido possível a composição de um ambiente distribuído com múltiplos nós, o presente trabalho não cobre a influência que o acréscimo e o decréscimo de nós ao cluster trazem para o desempenho geral dos bancos de dados e, uma análise mais aprofundada sobre este ponto, pode ajudar a responder questionamentos referentes aos limites do ambiente distribuído.

Outro tópico de interesse a trabalhos futuros é a análise metódica sobre a resiliência do cluster de SBCs, no intuito de investigar se de fato a maior suscetividade a falhas dos dispositivos utilizados pode ser um fator crítico ao funcionamento do cluster e se mesmo com as falhas o ambiente distribuído é capaz de se recuperar e manter-se operante.

Conforme observado com os resultados, as limitações de hardware dos dispositivos utilizados penalizam significativamente o desempenho dos bancos de dados em alguns cenários, impedindo um melhor aproveitamento de alguns recursos que poderiam possibilitar melhores resultados de desempenho. Sendo assim, a utilização de dispositivos dotados de melhores especificações de hardware, como o Raspberry Pi 4, podem trazer resultados consideravelmente melhores em experimentações futuras, além de permitir que uma maior gama de configurações dos bancos de dados seja explorada.

REFERÊNCIAS BIBLIOGRÁFICAS

ABRAMOVA, V.; BERNARDINO, J. Nosql databases: Mongodb vs cassandra. In: **Proceedings of the International C* Conference on Computer Science and Software Engineering**. New York, NY, USA: Association for Computing Machinery, 2013. (C3S2E '13), p. 14–22. ISBN 9781450319768.

ABRAMOVA, V.; BERNARDINO, J.; FURTADO, P. Evaluating cassandra scalability with ycsb. In: DECKER, H. et al. (Ed.). **Database and Expert Systems Applications**. Cham: Springer International Publishing, 2014. p. 199–207. ISBN 978-3-319-10085-2.

APACHE. **Apache Cassandra**. 2022. Acesso em 12 fev. 2022. Disponível em: <https://cassandra.apache.org/_/index.html>.

Apache Cassandra. **Dataset Partitioning: Consistent Hashing**: Consistent hashing using a token ring. 2016. Acesso em 18 dez. 2020. Disponível em: <https://cassandra.apache.org/doc/latest/architecture/dynamo.html#dataset-partitioning-consistent-hashing>.

____. **Dynamo**: Distributed cluster membership and failure detection. 2016. Acesso em 17 dez. 2020. Disponível em: <https://cassandra.apache.org/doc/latest/architecture/dynamo.html>.

APACHE HBASE. **Apache HBase Reference Guide**: Hbase run modes: Standalone and distributed. [S.l.], 2021. Acesso em 07 jan. 2021. Disponível em: <https://hbase.apache.org/book.html#standalone_dist>.

____. **Apache HBase Reference Guide**: Quick start - standalone hbase. [S.l.], 2021. Acesso em 07 jan. 2021. Disponível em: <https://hbase.apache.org/book.html#quickstart>.

ASHARI, A.; RIASETIAWAN, M. High performance computing on cluster and multicore architecture. **Telkomnika (Telecommunication Computing Electronics and Control)**, v. 13, n. 4, p. 1408–1413, 2015. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-84979737434&doi=10.12928%2fTELKOMNIKA.v13i4.2156&partnerID=40&md5=ad42a69870034ffae61d03a79a3fef2c>.

AWS. **Amazon DynamoDB**. 2022. Acesso em 12 fev. 2022. Disponível em: <https://aws.amazon.com/pt/dynamodb/>.

Big Data Europe. **Big Data Europe**: Integrating big data, software and communicaties for addressing europe societal challenges. 2020. Acesso em 15 dez. 2020. Disponível em: <https://github.com/big-data-europe>.

BOURHNANE, S. et al. Towards green data centers. In: AFONSO, J. L.; MONTEIRO, V.; PINTO, J. G. (Ed.). **Sustainable Energy for Smart Cities**. Cham: Springer International Publishing, 2020. p. 291–307. ISBN 978-3-030-45694-8.

BREWER, E. Cap twelve years later: How the "rules"have changed. **Computer**, v. 45, n. 2, p. 23–29, 2012.

BREWER, E. A. Towards robust distributed systems (abstract). In: **Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing**. New York, NY, USA: Association for Computing Machinery, 2000. (PODC '00), p. 7. ISBN 1581131836. Disponível em: <https://doi.org/10.1145/343477.343502>.

CHANG, F. et al. Bigtable: A distributed storage system for structured data. **ACM Trans. Comput. Syst.**, Association for Computing Machinery, New York, NY, USA, v. 26, n. 2, jun 2008. ISSN 0734-2071.

Citus Data. **Citus Data**: Scalable postgresql. 2021. Acesso em 20 fev. 2021. Disponível em: <https://github.com/citusdata>.

____. **membership-manager**. 2021. Acesso em 23 fev. 2021. Disponível em: <https://github.com/citusdata/membership-manager>.

CitusData. **Concepts**. 2022. <https://docs.citusdata.com/en/v10.2/get_started/concepts.html>. Accessed: Fev 26, 2022.

COOPER, B. F. et al. Benchmarking cloud serving systems with ycsb. In: **Proceedings of the 1st ACM Symposium on Cloud Computing**. New York, NY, USA: Association for Computing Machinery, 2010. (SoCC '10), p. 143–154. ISBN 9781450300360.

CUBUKCU, U. et al. Citus: Distributed postgresql for data-intensive applications. In: **Proceedings of the 2021 International Conference on Management of Data**. New York, NY, USA: ACM, 2021. (SIGMOD '21), p. 2490–2502. ISBN 9781450383431. Disponível em: <https://doi.org/10.1145/3448016.3457551>.

DATASTAX. **Benchmarking Top NoSQL Databases**. [S.l.], 2015.

DAVOUDIAN, A.; CHEN, L.; LIU, M. A survey on nosql stores. **ACM Comput. Surv.**, ACM, New York, NY, USA, v. 51, n. 2, apr 2018. ISSN 0360-0300. Disponível em: <https://doi.org/10.1145/3158661>.

DEB, S. K. **Docker With Spring Boot and MySQL: Docker Swarm Part 3**. 2021. Acesso em 10 jan. 2021. Disponível em: <https://dzone.com/articles/docker-with-spring-boot-and-mysql-docker-swarm-par>.

DECANDIA, G. et al. Dynamo: Amazon's highly available key-value store. In: **Proceedings of Twenty-First ACM SIGOPS Symposium on Operating Systems Principles**. New York, NY, USA: Association for Computing Machinery, 2007. p. 205–220. ISBN 9781595935915.

DOCKER. **What is a Container?** 2021. Acesso em 16 jan. 2021. Disponível em: <https://www.docker.com/resources/what-container>.

____. **What is a swarm?** 2021. Acesso em 10 jan. 2021. Disponível em: <https://docs.docker.com/engine/swarm/key-concepts/#what-is-a-swarm>.

Docker Community. **Docker Official Image for Cassandra**. 2020. Acesso em 20 out. 2020. Disponível em: <https://github.com/docker-library/cassandra>.

ELMASRI, R.; NAVATHE, S. B. **Fundamentals of Database Systems**. 7th. ed. [S.l.]: Pearson, 2015. ISBN 0133970779.

FAYOS-JORDAN, R. et al. Elastic computing in the fog on internet of things to improve the performance of low cost nodes. **Electronics**, v. 8, n. 12, 2019. ISSN 2079-9292. Disponível em: <https://www.mdpi.com/2079-9292/8/12/1489>.

FIGUIèRE, M. **Cassandra error handling done right**. 2014. <https://www.datastax.com/blog/2014/10/cassandra-error-handling-done-right>. Accessed: Jul 09, 2020.

GILBERT, S.; LYNCH, N. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. **SIGACT News**, Association for Computing Machinery, New York, NY, USA, v. 33, n. 2, p. 51–59, jun 2002. ISSN 0163-5700. Disponível em: <https://doi.org/10.1145/564585.564601>.

GROUP, T. P. G. D. **PostgreSQL: The World's Most Advanced Open Source Relational Database**. 2022. Acesso em 12 fev. 2022. Disponível em: <https://www.postgresql.org/>.

HADOOP, A. **Apache Hadoop**. 2006. Acesso em 15 jan. 2021. Disponível em: <http://hadoop.apache.org/>.

HENDAWI, A. et al. Benchmarking large-scale data management for internet of things. **The Journal of Supercomputing**, v. 75, n. 12, p. 8207–8230, Dec 2019. ISSN 1573-0484.

HUANG, X. et al. Optimizing data partition for scaling out nosql cluster. **Concurrency and Computation: Practice and Experience**, v. 27, n. 18, p. 5793–5809, 2015.

IBM. **IBM InfoSphere BigInsights Version 3.0**: Hbase basics. [S.l.], 2014. Acesso em 07 jan. 2021. Disponível em: <https://www.ibm.com/support/knowledgecenter/SSPT3X_3.0.0/com.ibm.swg.im.infosphere.biginsights.analyze.doc/doc/hbaseConcepts.html>.

IBM Cloud Education. **Teorema CAP**. 2019. <https://www.ibm.com/br-pt/cloud/learn/cap-theorem>. Accessed: Fev 22, 2022.

JOHNSTON, S. J. et al. Commodity single board computer clusters and their applications. **Future Generation Computer Systems**, v. 89, p. 201 – 212, 2018. ISSN 0167-739X. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167739X18301833>.

JUBA, S.; VOLKOV, A. **Learning PostgreSQL 11**. Birmingham: Packt Publishing Ltd, 2019. 529 p.

KIM, H.-J. et al. Techniques and guidelines for effective migration from rdbms to nosql. **The Journal of Supercomputing**, v. 76, n. 10, p. 7936–7950, Oct 2020. ISSN 1573-0484. Disponível em: <https://doi.org/10.1007/s11227-018-2361-2>.

KUHLENKAMP, J.; KLEMS, M.; RöSS, O. Benchmarking scalability and elasticity of distributed database systems. **Proc. VLDB Endow.**, VLDB Endowment, v. 7, n. 12, p. 1219–1230, ago. 2014. ISSN 2150-8097.

LAKSHMAN, A.; MALIK, P. Cassandra: A decentralized structured storage system. **SIGOPS Oper. Syst. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 44, n. 2, p. 35–40, 2010. ISSN 0163-5980.

MORABITO, R.; KJäLLMAN, J.; KOMU, M. Hypervisors vs. lightweight virtualization: A performance comparison. In: . [S.l.: s.n.], 2015.

NUTTALL, B. **What is a Raspberry Pi?** 2019. Acesso em 21 jan. 2021. Disponível em: <https://opensource.com/resources/raspberry-pi>.

Oracle Corporation. **MySQL**. 2022. Acesso em 12 fev. 2022. Disponível em: <https://www.mysql.com/>.

PAPAKYRIAKOU, D. Benchmarking raspberry pi 2 hadoop cluster. **International Journal of Computer Applications**, v. 178, p. 37–47, 08 2019.

PAPAPANAGIOTOU, I.; CHELLA, V. **NDBench: Benchmarking Microservices at Scale**. 2018.

PRIYAMBODO, T.; LISAN, A.; RIASETIAWAN, M. Inexpensive green mini supercomputer based on single board computer cluster. **Journal of Telecommunication, Electronic and Computer Engineering**, v. 10, n. 1-6, p. 141–145, 2018. Disponível em: <https://www.scopus.com/inward/record.uri?eid=2-s2.0-85045180724&partnerID=40&md5=2d4be2001f4e7f25a07e9aa22f5b1a6e>.

Raspberry Pi Foundation. **Raspberry Pi 4 Tech Specs**. 2020. Acesso em 21 jan. 2021. Disponível em: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>.

Redis Ltd. **Redis**. 2022. Acesso em 12 fev. 2022. Disponível em: <https://redis.io/>.

REINSEL JOHN GANTZ, J. R. D. **The Digitization of the World From Edge to Core**. 2018. <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf>. Accessed: Jul 07, 2020.

Richardson, D. P.; Lin, A. C.; Pecarina, J. M. Hosting distributed databases on internet of things-scale devices. In: **2017 IEEE Conference on Dependable and Secure Computing**. [S.l.: s.n.], 2017. p. 352–357.

SCHöNIG, H.-J. **PostgreSQL Replication**. Birmingham: Packt Publishing Ltd, 2015. 293 p.

SHVACHKO, K. et al. The hadoop distributed file system. In: . USA: IEEE Computer Society, 2010. (MSST '10), p. 1–10. ISBN 9781424471522. Disponível em: <https://doi.org/10.1109/MSST.2010.5496972>.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Database System Concepts, Seventh Edition**. [S.l.]: McGraw-Hill Book Company, 2020.

SILVA, L. F. D.; LIMA, J. V. F. An evaluation of cassandra nosql database on a low-power cluster. In: **2021 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)**. [S.l.: s.n.], 2021. p. 9–14.

STEFFENEL., L. A.; CHARãO., A. S.; ALVES., B. da S. A containerized tool to deploy scientific applications over soc-based systems: The case of meteorological forecasting with wrf. In: INSTICC. **Proceedings of the 9th International Conference on Cloud Computing and Services Science - Volume 1: CLOSER,**. [S.l.]: SciTePress, 2019. p. 561–568. ISBN 978-989-758-365-0.

Stonebraker, M.; Rowe, L. A.; Hirohama, M. The implementation of postgres. **IEEE Transactions on Knowledge and Data Engineering**, v. 2, n. 1, p. 125–142, 1990.

Swaminathan, S. N.; Elmasri, R. Quantitative analysis of scalable nosql databases. In: **2016 IEEE International Congress on Big Data (BigData Congress)**. [S.l.: s.n.], 2016. p. 323–326.

Tang, E.; Fan, Y. Performance comparison between five nosql databases. In: **2016 7th International Conference on Cloud Computing and Big Data (CCBD)**. [S.l.: s.n.], 2016. p. 105–109.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP. **PostgreSQL 12.6 Documentation**: Architectural fundamentals. [S.l.], 2021. Acesso em 15 fev. 2021. Disponível em: <https://www.postgresql.org/docs/12/tutorial-arch.html>.

WITTE, R. S.; WITTE, J. S. **Statistics**. [S.l.]: Wiley, 2017.

WOLF, W.; JERRAYA, A. A.; MARTIN, G. Multiprocessor system-on-chip (mpsoc) technology. **Trans. Comp.-Aided Des. Integ. Cir. Sys.**, IEEE Press, Piscataway, NJ, USA, v. 27, n. 10, p. 1701–1713, out. 2008. ISSN 0278-0070. Disponível em: <https://doi.org/10.1109/TCAD.2008.923415>.

ZOOKEEPER, A. **Welcome to Apache ZooKeeper**. 2010. Acesso em 15 jan. 2021. Disponível em: <https://zookeeper.apache.org/>.

APÊNDICE A – MÉDIAS DE USO DE CPU DE CADA RPI DO CLUSTER PARA CADA WORKLOAD

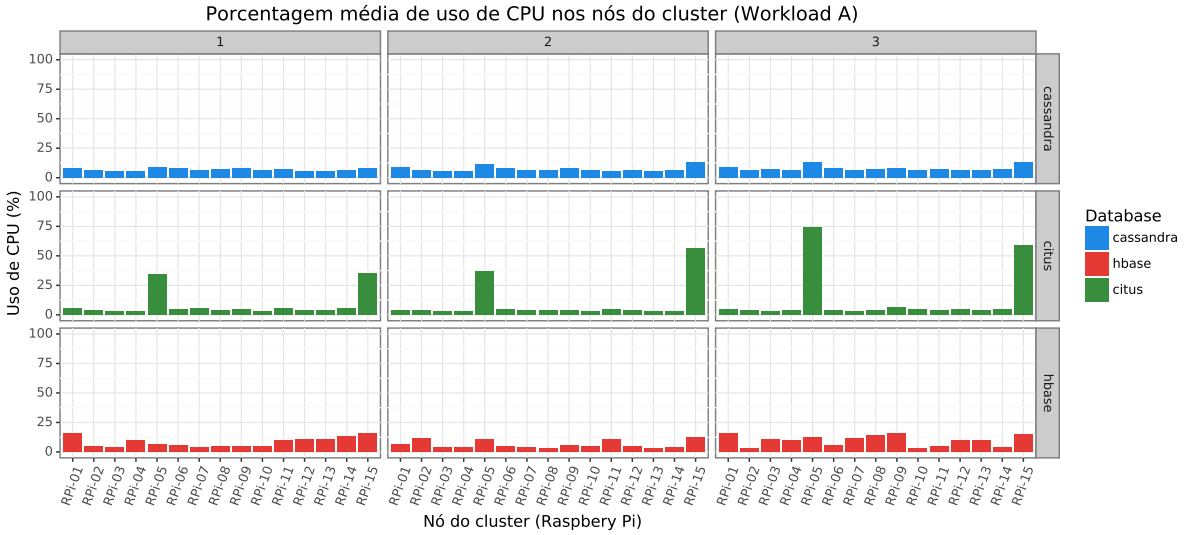


Gráfico A.1 – Médias da porcentagem de uso de CPU de cada nó do cluster durante uma das 30 execuções do workload A

Fonte: O autor.

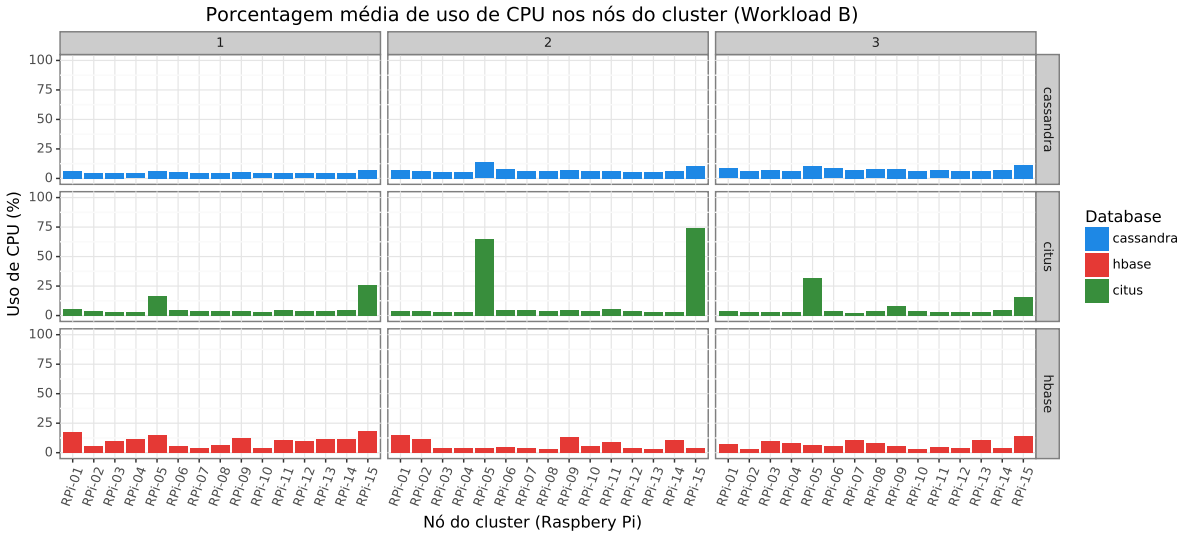


Gráfico A.2 – Médias da porcentagem de uso de CPU de cada nó do cluster durante uma das 30 execuções do workload B

Fonte: O autor.

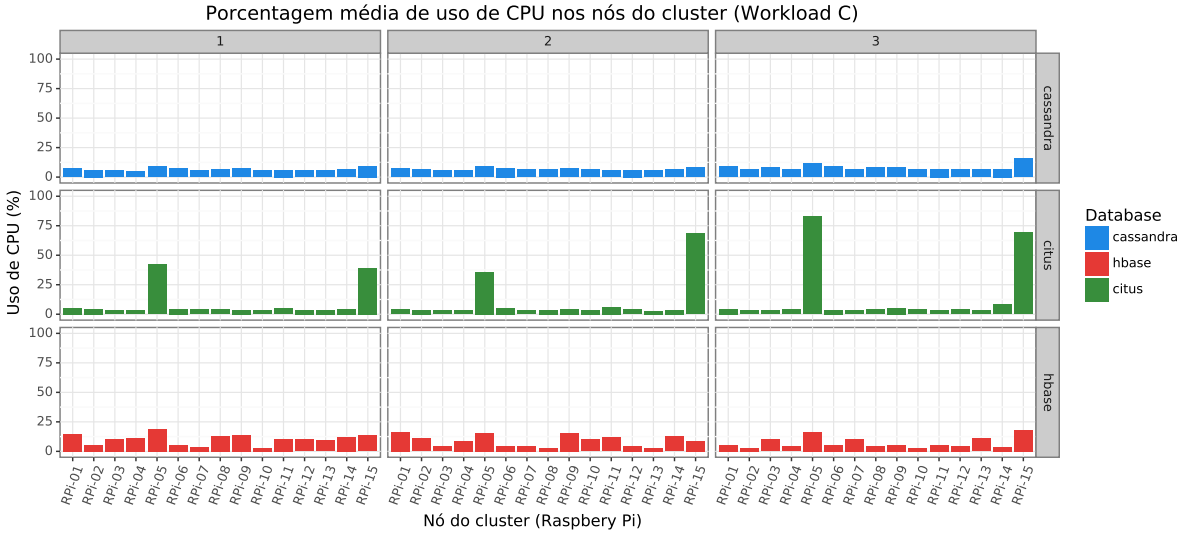


Gráfico A.3 – Médias da porcentagem de uso de CPU de cada nó do cluster durante uma das 30 execuções do workload C

Fonte: O autor.

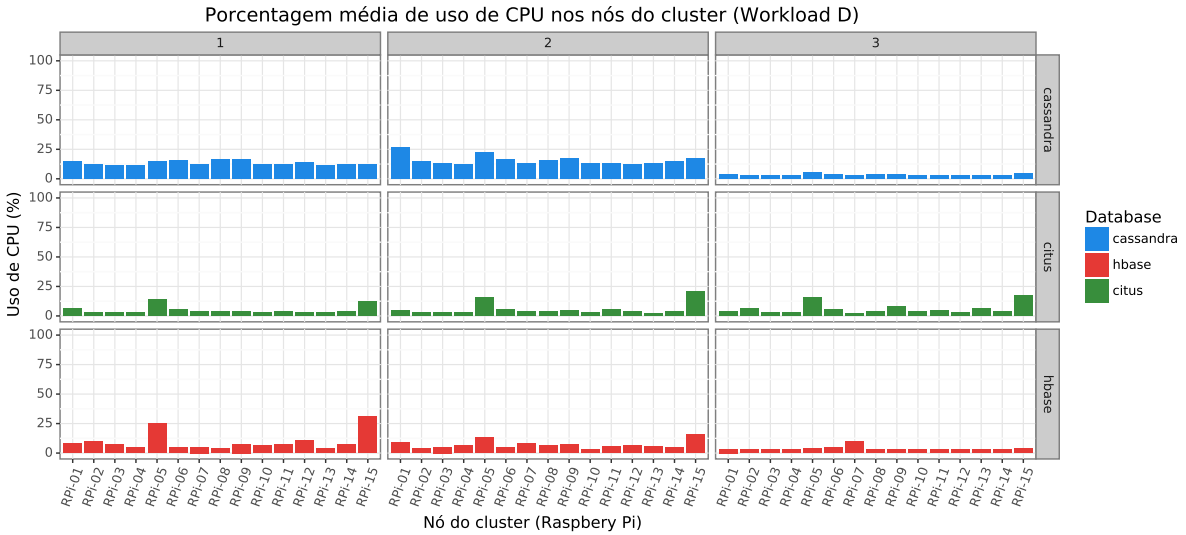


Gráfico A.4 – Médias da porcentagem de uso de CPU de cada nó do cluster durante uma das 30 execuções do workload D

Fonte: O autor.