

UNIVERSIDADE FEDERAL DO PARANÁ  
SETOR DE CIÊNCIAS EXATAS  
CURSO DE CIÊNCIA DA COMPUTAÇÃO

MARIA TERESA KRAVETZ ANDRIOLI

OTIMIZAÇÃO DO HADOOP MAPREDUCE ATRAVÉS DO TUNING DOS PARÂMETROS  
DE CONFIGURAÇÃO

CURITIBA

2022

MARIA TERESA KRAVETZ ANDRIOLI

OTIMIZAÇÃO DO HADOOP MAPREDUCE ATRAVÉS DO TUNING DOS PARÂMETROS  
DE CONFIGURAÇÃO

Trabalho apresentado como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação no curso de Ciência da Computação, Setor de Ciências Exatas da Universidade Federal do Paraná.

Orientador: Prof. Dr. Luiz Carlos P. Albini

CURITIBA

2022

## **TERMO DE APROVAÇÃO**

**MARIA TERESA KRAVETZ ANDRIOLI**

### **OTIMIZAÇÃO DO HADOOP MAPREDUCE ATRAVÉS DO TUNING DOS PARÂMETROS DE CONFIGURAÇÃO**

Trabalho apresentado como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação no curso de Ciência da Computação, Setor de Ciências Exatas da Universidade Federal do Paraná, pela seguinte banca examinadora:

---

**Prof. Dr. Luiz Carlos P. Albini**  
**Orientador**

---

Professora  
UFPR

---

Professora

---

Professora

Curitiba, Maio de 2022.

## **AGRADECIMENTOS**

ADICIONAR AGRADECIMENTOS

*“Tudo que posso fazer é aceitar o pandemônio.  
Encontrar felicidade na insanidade única de estar aqui agora.”  
(Amram e Statsky (2019) - Pandemonium, The Good Place)*

## RESUMO

Um dos grandes desafios da área de computação sempre foi lidar com o uso, armazenamento e controle de dados. *Big Data* refere-se a grandes aglomerados de dados os quais ferramentas tradicionais não conseguem processar de forma otimizada. Assim, foi desenvolvido pelo Google o paradigma de programação *MapReduce*, com o objetivo de simplificar esse processo. O *Hadoop* é um *framework* criado pela Apache para armazenar e processar em paralelo grandes quantidades de dados. Em conjunto, eles formam o *Hadoop MapReduce*, que agrega os benefícios das duas ferramentas a fim de obter uma execução otimizada.

O *Hadoop MapReduce* contém uma grande quantidade de parâmetros de configuração que podem - e devem - ser alterados com base na situação sobre o qual ele está sendo aplicado, o ambiente no qual ele está sendo executado e os objetivos a serem atingidos pela aplicação. A alteração desses parâmetros com objetivo de melhora performance é chamada de *tuning* e a boa manipulação desses valores é essencial para que o *framework* funcione da melhor maneira possível.

Dessa forma, o presente trabalho apresenta as ferramentas mencionadas, os valores iniciais dos parâmetros e os experimentos realizados e resultados obtidos após o *tuning* das configurações do *Hadoop MapReduce*.

**Palavras-chaves:** Big Data; Hadoop; MapReduce; otimização; parâmetros.

## ABSTRACT

One of the biggest challenges of the computing area has always been dealing with data use, storage and manipulation. Big Data is the term that defines big conglomerates of data which cannot be processed by traditional tools in an optimized way. With that in mind, Google has developed the programming paradigm called MapReduce to simplify this process. Hadoop is a framework created by Apache to store and process big amount of data in parallel. Together, they make Hadoop MapReduce, which aggregates the benefits from both tools in order to obtain an optimized performance.

Hadoop MapReduce has a vast amount of configuration parameters that can - and should - be altered according to the application, the execution environment and which goals are to be achieved after your execution. Altering those parameters in order to have an improvement in performance is called tuning and handling those values well is essential for the framework to work in the best way possible.

Thereby, the present work presents the aforementioned tools, the default values for the parameters and the experiments and results obtained after tuning Hadoop Map Reduce configuration.

**Key-words:** Big Data; Hadoop; MapReduce; optimization; framework; tuning; parameters.

## LISTA DE CÓDIGOS

|     |   |    |
|-----|---|----|
| 2.1 | Exemplo de função Map em pseudocódigo adaptado de (DEAN; GHEMAWAT, 2008) . . . . .    | 14 |
| 2.2 | Exemplo de função Reduce em pseudocódigo adaptado de (DEAN; GHEMAWAT, 2008) . . . . . | 14 |
| 3.1 | Exemplo de execução do <i>TeraGen</i> adaptado de (WHITE, 2015) . . . . .             | 25 |
| 3.2 | Exemplo de execução do <i>TeraSort</i> adaptado de (WHITE, 2015) . . . . .            | 25 |
| 3.3 | Exemplo de execução do <i>TeraValidate</i> adaptado de (WHITE, 2015) . . . . .        | 26 |



## LISTA DE ILUSTRAÇÕES

|  |    |
|--|----|
| FIGURA 1 – EXECUÇÃO GENÉRICA DO MAPREDUCE . . . . .                          | 15 |
| FIGURA 2 – EXEMPLO DE EXECUÇÃO DO MAPREDUCE . . . . .                        | 16 |
| FIGURA 3 – NOVA ARQUITETURA DO HADOOP 2.0 . . . . .                          | 19 |
| FIGURA 4 – ARQUITETURA DOCKER . . . . .                                      | 19 |
| FIGURA 5 – EXECUÇÃO DO <i>TERASORT</i> COM PARÂMETROS DE <i>TUNING</i> . . . | 26 |
| FIGURA 6 – RESULTADOS DO <i>TUNING</i> . . . . .                             | 29 |

## LISTA DE QUADROS

|   |    |
|---|----|
| QUADRO 1 – FATORES PARA <i>TUNING</i> DO <i>JOB MAPREDUCE</i> . . . . . | 21 |
| QUADRO 2 – PARÂMETROS DE AJUSTE DA QUANTIDADE DE TAREFAS <i>MAP</i> .   | 23 |
| QUADRO 3 – PARÂMETROS DE AJUSTE DA QUANTIDADE DE TAREFAS <i>REDUCE</i>  | 24 |
| QUADRO 4 – PARÂMETROS ADICIONAIS DE CONFIGURAÇÃO . . . . .              | 24 |
| QUADRO 5 – PARÂMETROS AJUSTADOS DURANTE O <i>TUNING</i> . . . . .       | 28 |
| QUADRO 6 – PARÂMETROS NÃO AJUSTADOS DURANTE O <i>TUNING</i> . . . . .   | 28 |

## SUMÁRIO

|          |                                |           |
|----------|--------------------------------|-----------|
| <b>1</b> | <b>INTRODUÇÃO</b>              | <b>11</b> |
| 1.1      | CONTEXTO                       | 11        |
| 1.2      | OBJETIVO                       | 12        |
| 1.3      | ESTRUTURA DO TRABALHO          | 12        |
| <b>2</b> | <b>REFERENCIAL TEÓRICO</b>     | <b>13</b> |
| 2.1      | CLUSTERS                       | 13        |
| 2.2      | MAPREDUCE                      | 13        |
| 2.2.1    | Modelo de programação          | 14        |
| 2.2.2    | Execução do MapReduce          | 15        |
| 2.3      | HADOOP                         | 16        |
| 2.3.1    | Hadoop Common                  | 17        |
| 2.3.2    | Hadoop HDFS                    | 17        |
| 2.3.3    | Hadoop MapReduce               | 18        |
| 2.3.4    | Hadoop YARN                    | 18        |
| 2.4      | VIRTUALIZAÇÃO                  | 19        |
| <b>3</b> | <b>OTIMIZAÇÃO DO MAPREDUCE</b> | <b>21</b> |
| 3.1      | PARÂMETROS DO <i>MAPREDUCE</i> | 21        |
| 3.2      | TERASORT                       | 25        |
| 3.3      | AMBIENTE EXPERIMENTAL          | 26        |
| 3.4      | RESULTADOS                     | 27        |
| <b>4</b> | <b>CONCLUSÃO</b>               | <b>30</b> |
|          | <b>REFERÊNCIAS</b>             | <b>31</b> |
|          | <b>GLOSSÁRIO</b>               | <b>34</b> |

# 1 INTRODUÇÃO

## 1.1 CONTEXTO

O uso, armazenamento e controle de dados é um tema muito discutido na área de computação desde seus primórdios até os dias de hoje. Dessa forma, muitos métodos e algoritmos e termos surgiram ao longo do tempo com o objetivo de gerenciar de forma eficiente esses dados. O surgimento de tais ferramentas computacionais e métodos de armazenamento é de grande importância para a evolução da área.

Atualmente, os métodos mais comuns são bancos de dados relacionais e *data warehouses* usando computação em nuvem (KUO; KUSIAK, 2019). Além disso, pesquisas nos campos de mineração de dados e aprendizagem de máquina cresceram bastante recentemente, de modo a prover técnicas que permitissem analisar dados complexos e variados entre si (BELCASTRO et al., 2022). Um grande desafio é o fato de algoritmos sequenciais não serem otimizados o suficiente para lidar com dados em grande quantidade. Assim, computadores de alta performance, com múltiplos *cores*, sistemas na nuvem e algoritmos paralelos e distribuídos são usados para lidar com esses empecilhos de *Big Data* (BELCASTRO et al., 2022).

*Big Data* refere-se a grandes conglomerados de dados complexos sobre os quais não é possível aplicar ferramentas tradicionais de processamento, armazenamento ou análise (KHALEEL; AL-RAWESHIDY, 2018). Estima-se que em 2025 os dados atuais criados, capturados ou replicados atinjam 175 Zettabytes, ou seja 175.000.000.000 Gigabytes (RYDNING, 2018).

A fim de lidar com essa enorme quantidade de dados, foi desenvolvido pelo Google o *MapReduce*, um modelo de programação com uma implementação associada feito para processar e gerar grandes conglomerados de dados. Esse modelo é inspirado nos conceitos de mapear e reduzir, ou seja, aplicar uma operação que conecta cada item da base de dados a um determinado par de chaves e valores, e então aplicar uma operação de reduzir, que une os valores que compartilham chaves (DEAN; GHEMAWAT, 2008). Com essas operações é possível paralelizar dados em grandes quantidades e utilizar mecanismos de reutilização para facilitar a busca e a manipulação destes.

Um dos *frameworks* mais populares que utiliza o *MapReduce* é o *Hadoop*, desenvolvido pela Apache em 2006 e capaz de armazenar e processar de Gigabytes a Petabytes de dados eficientemente, optando por usar múltiplos computadores (*clusters*) em paralelo (WHITE, 2015).

## 1.2 OBJETIVO

O *Hadoop MapReduce* é um *framework* extremamente personalizável e adaptável. Dessa forma, frequentemente utiliza-se o processo de *tuning*, que consiste em modificar os mais de 190 parâmetros desse *framework* de modo a maximizar a eficiência de um *cluster Hadoop*. Esses parâmetros podem ser alterados em diversas combinações e podem ter efeitos tanto no *cluster* quanto nas tarefas (*jobs*) do processo.

Esse trabalho tem como objetivo avaliar o comportamento do *Hadoop MapReduce* antes e depois do *tuning* de alguns parâmetros de configuração, observando através de métricas de *benchmark* se houve melhora na performance, considerando medidas como tempo e uso de memória.

## 1.3 ESTRUTURA DO TRABALHO

Esse trabalho está organizado da seguinte forma: o Capítulo 2 apresenta os fundamentos teóricos e as ferramentas que serão utilizadas nos experimentos e referenciadas durante o desenvolvimento do texto; o Capítulo 3 apresenta em detalhes os parâmetros do *Hadoop MapReduce*, as ferramentas de *benchmark* utilizadas para obtenção das métricas, o ambiente no qual os experimentos serão executados e os resultados obtidos no através dos ajustes dos parâmetros de configuração da ferramenta. Finalmente, no Capítulo 4 é feita a conclusão do trabalho.

## 2 REFERENCIAL TEÓRICO

Este capítulo tem como objetivo apresentar detalhadamente os conceitos técnicos que serão utilizados ao longo do trabalho. A seção 2.1 introduz o conceito de *clusters*. A seção 2.2 apresenta o *MapReduce*, o modelo de manipulação de dados feito pelo Google e a seção 2.3 trata do *Hadoop*, o *framework* desenvolvido pela Apache. Por fim, a seção 2.4 explica virtualização, contêineres e a ferramenta Docker.

### 2.1 CLUSTERS

Um *cluster* é um conjunto de computadores que trabalham juntos paralelamente em uma determinada aplicação. Cada computador desse conjunto é usualmente chamado de nodo. Além disso, existem diversas categorias de *clusters*, dependendo do problema que eles buscam computar (GOLDMAN et al., 2012).

Algumas aplicações comuns de *clusters* são modelagem de clima, simulação de acidentes automotivos, mineração de dados e aplicações da área de astrofísica. Além disso, é comumente visto em aplicações comerciais como bancos e serviços de email (SADASHIV; KUMAR, 2011).

Uma das maiores vantagens desse tipo de instalação é a tolerância de falhas, pois os sistemas conseguem continuar suas tarefas caso um nodo pare de funcionar. Além disso, é altamente escalável com a adição de novos nodos, não precisa de manutenção frequente e tem um gerenciamento centralizado. Por fim, uma das suas maiores possíveis vantagens é o balanceamento de carga, que busca atingir o equilíbrio entre as tarefas de cada nodo de modo a otimizar os recursos (SADASHIV; KUMAR, 2011).

### 2.2 MAPREDUCE

*MapReduce* é um modelo de programação associado a uma implementação que tem como objetivo processar, manipular e gerar grandes *datasets* de modo eficiente, escalável e com aplicações no mundo real. As computações acontecem de acordo com funções de mapeamento e redução e o sistema do *MapReduce* paraleliza essas computações entre grandes *clusters*, lidando com possíveis falhas, escalonamentos e uso eficiente de rede e discos (DEAN; GHEMAWAT, 2008).

As operações de mapeamento e redução são baseadas em conceitos presentes em linguagens funcionais e fazem com que seja possível fazer diversas reutilizações, assim lidando com tolerância de falhas (DEAN; GHEMAWAT, 2008).

### 2.2.1 Modelo de programação

A computação recebe um conjunto de pares (CHAVE, VALOR) e produz um conjunto de pares de (CHAVE, VALOR). O usuário cria as funções *Map* e *Reduce* de acordo com seu uso. *Map* recebe um único par (CHAVE, VALOR) e produz um conjunto intermediário de pares. Em seguida, a biblioteca *MapReduce* agrupa os valores com a mesma chave, os quais servirão de entrada para a função *Reduce*. Nesse momento a função *Reduce* unifica os valores com a mesma chave de modo a criar um conjunto menor, sendo possível dessa forma lidar com listas muito grandes para a memória disponível (DEAN; GHEMAWAT, 2008).

Entre o momento que são executadas as funções *Map* e *Reduce*, existe a fase *Shuffle*, que é criada automaticamente em tempo de execução e executa operações de ordenação (*sort*) e junção (*merge*) (VENNER, 2009). Para White (2015), a operação *Shuffle* é um dos fatores mais influentes no bom desempenho de aplicações *MapReduce*, uma vez que operações de ordenação e junções podem prejudicar ou melhorar muito um algoritmo conforme sua implementação.

Como exemplo, considere-se o problema de contar quantas vezes determinada palavra aparece em um documento. Nesse problema, as funções *Map* e *Reduce* seriam similares aos seguintes pseudocódigos (DEAN; GHEMAWAT, 2008):

---

```

1 map(String chave, String valor):
2   // chave: nome do documento
3   // valor: conteúdo do documento
4
5   para cada palavra W em valor:
6     criaIntermediario(W, 1);
```

---

CÓDIGO 2.1 – Exemplo de função Map em pseudocódigo adaptado de (DEAN; GHEMAWAT, 2008)

---

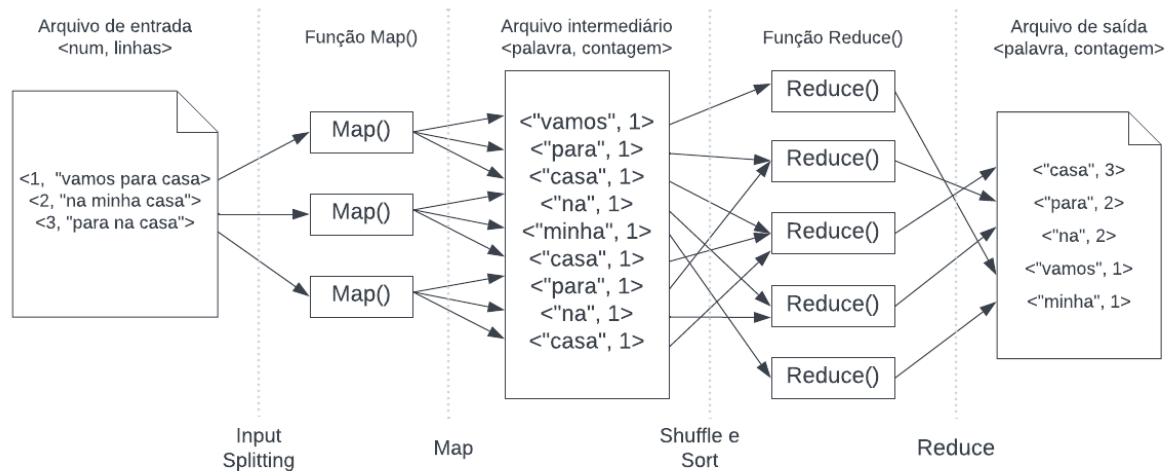
```

1 reduce(String chave, Iterador valores):
2   // chave: uma palavra
3   // valores: lista de contagens
4
5   int resultado = 0;
6   para cada V em valores:
7     resultado = resultado + 1;
8   cria(resultado);
```

---

CÓDIGO 2.2 – Exemplo de função Reduce em pseudocódigo adaptado de (DEAN; GHEMAWAT, 2008)

FIGURA 1 – EXECUÇÃO GENÉRICA DO MAPREDUCE



FONTE: A autora (2022)

A função *Map* gera um objeto intermediário de cada palavra associada a uma lista do seu número de ocorrências no texto e a função *Reduce* soma os valores até que essas ocorrências por palavras sejam totalizadas. Além disso, o usuário cria uma configuração de *MapReduce* com os parâmetros de entrada e saída e eventuais parâmetros de *tuning*.

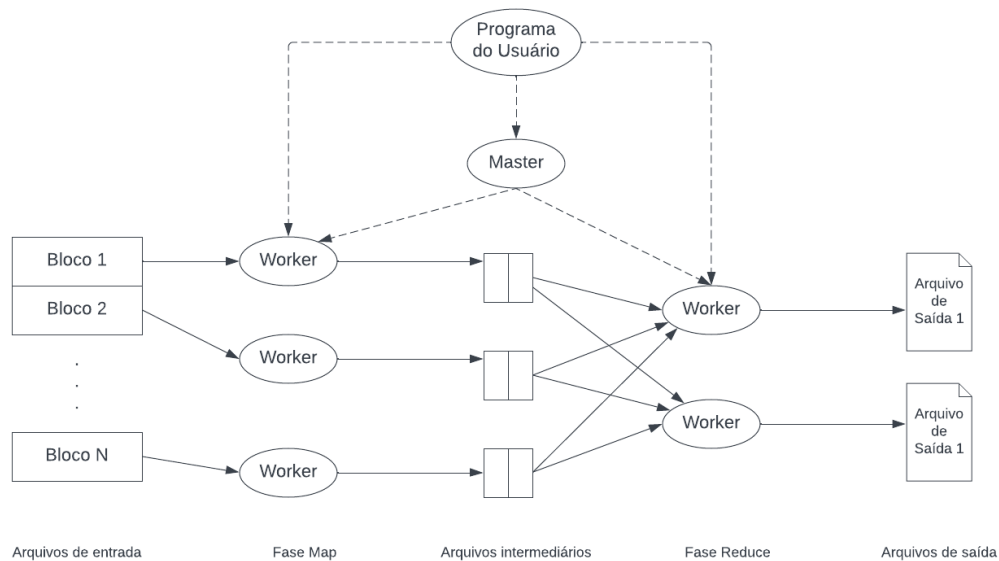
Para exemplificar ainda mais, considere um arquivo de texto com três linhas nas quais estão as seguintes frases, respectivamente, uma em cada linha: "vamos para casa", "na minha casa", "para na casa". Nesse exemplo, a função *Map* é chamada três vezes, uma para cada linha, gerando os pares (CHAVE, VALOR) intermediários, um para cada palavra encontrada no texto, como é mostrado na FIGURA 1. Para cada palavra distinta ("vamos", "para", "casa", "na", "minha"), é executada a função *Reduce*, que soma quantas vezes cada uma dessas palavras apareceu no texto e gera um arquivo de saída.

## 2.2.2 Execução do MapReduce

O *MapReduce* funciona usando uma estrutura Cliente/Servidor sobre um *cluster* que, segundo Dean e Ghemawat (2008), consiste em primeiramente particionar os dados de entrada em blocos de tamanho já definidos e depois distribuir cópias do programa MapReduce entre cada um desses blocos. Existe uma cópia Master, que é responsável por repartir as tarefas (*tasks*), enquanto as demais cópias - denominadas Workers - recebem da Master as tarefas junto com os arquivos de entrada. Ao finalizar a execução de uma tarefa do tipo *Map*, a cópia Worker responsável repassa a Master os arquivos de saída e esta repassa a um Worker esse arquivo com a tarefa de *Reduce*. Por fim, esse worker executa a redução, lendo os pares intermediários que passaram pela fase *Shuffle* e agrupando as instâncias de mesma chave. Quando todas as tarefas *Map* e *Reduce* forem executadas, o programa é finalizado.



FIGURA 2 – EXEMPLO DE EXECUÇÃO DO MAPREDUCE



FONTE: Adaptado de (DEAN; GHEMAWAT, 2008)

Na FIGURA 1 foi possível ver como o *MapReduce* funcionaria em pequena escala. Uma das maiores vantagens do *MapReduce* é, no entanto, sua escalabilidade, visto que ele permite uma execução distribuída entre uma grande quantidade de nodos. A FIGURA 2 representa uma execução genérica do *MapReduce*, descrita no parágrafo acima.

### 2.3 HADOOP

*Hadoop* é um *framework* desenvolvido na linguagem Java pela Apache Software Foundation com os seguintes princípios arquiteturais, segundo Navarro Belmonte (2018):

- A possibilidade de escalar o sistema ao adicionar nodos no *cluster*.
- Possibilidade de funcionar bem em *hardware* que não necessite ser caro e de luxo.
- Tolerância a falhas, com implementações que as identificam e permitem que o sistema funcione independente delas acontecerem.
- Fornecimento de serviços para que o usuário foque no problema que deseja resolver.

Esse *framework* disponibiliza ferramentas para que o usuário possa escrever as funções necessárias em diversas linguagens de programação, conforme a necessidade do programador. O *framework* funciona na mesma estrutura de Cliente/Servidor apresentada anteriormente, utilizada pelo *MapReduce*. Além disso, oferece ao programador um sistema paralelo e distribuído (*Hadoop HDFS*), com os recursos ocultos ao usuário, mas capaz de lidar com a comunicação entre as máquinas e quaisquer falhas que possam vir a ocorrer e o escalonamento das tarefas.

Além do *Hadoop Map Reduce* e do *Hadoop HDFS*, existem outros subprojetos do Hadoop que compõem sua estrutura principal: o *Hadoop Common*, que fornece ferramentas comuns aos outros subprojetos o *Hadoop YARN*, um *framework* para escalonamento de tarefas e gerenciamento de recursos em *clusters*.

### 2.3.1 Hadoop Common

Esse subprojeto contém os utilitários e bibliotecas comuns aos outros subprojetos. Por exemplo, funções de manipulação de arquivos, funções auxiliares de serialização de dados, etc (GOLDMAN et al., 2012).

### 2.3.2 Hadoop HDFS

Segundo HDFS... (2020) o *Hadoop HDFS* é um sistema de arquivos distribuídos criado para funcionar em *hardware* facilmente obtido e relativamente barato. Suas características principais são a alta capacidade de lidar com falhas e a possibilidade de ser usado com aplicações que possuem grande quantidades de dados como entrada.

Uma instância HDFS é composta de centenas ou milhares de máquinas, cada uma responsável por armazenar uma parte dos dados do sistema. Dessa forma, a rápida detecção e recuperação de falhas é essencial para sua estrutura. Seu *design* foi pensado em aplicações de processamento de dados em blocos e o tamanho de seus arquivos pode variar entre Giga e Terabytes. Além disso, é adaptado para funcionar em diferentes plataformas e prover interfaces que possibilitam mover a aplicação para perto dos dados, permitindo que qualquer operação computacional aplicada seja muito mais eficiente (HDFS... , 2020).

O *Hadoop HDFS* também possui uma estrutura Cliente/Servidor, em que o *Namenode* - responsável por gerenciar o sistema e regular o acessos aos arquivos - é o nodo Master e os *Datanodes* - responsáveis por gerenciar o armazenamento dos nodos aos quais eles estão conectados - são os nodos Worker. O sistema é implementado usando uma estrutura comum de diretórios na qual é possível criar, mover, renomear e remover arquivos, mas ainda não implementa funções como quota de usuários, permissões de acesso ou *links* simbólicos (HDFS... , 2020).

Uma das características essenciais desse sistema é sua capacidade de lidar com grandes quantidades de dados. Segundo HDFS... (2020), isso é feito através do armazenamento dos arquivos como uma sequência de blocos, cujo tamanho e fator de replicação são configuráveis pelo usuário, mas possuem um valor padrão de 64MB. Periodicamente, o *Namenode* recebe dos *Datanodes* um sinal indicando se o funcionamento está correto. O posicionamento e a quantidade de réplicas de um bloco é crítica na análise da boa performance do HDFS, e é um dos fatores que o diferencia de outros sistemas de arquivos distribuídos. Quando executada de forma otimizada, pode aumentar confiabilidade, disponibilidade e uso de redes do sistema.

### 2.3.3 Hadoop MapReduce

O *Hadoop MapReduce* é um *framework* que implementa o modelo de programação *MapReduce* para facilitar a criação de aplicações que são capazes de processar grandes quantidades de dados em paralelo em *clusters* de uma forma confiável e com tolerância a falhas.

Esse *framework* é constituído de um *Job* responsável por dividir os arquivos de entrada em blocos independentes que serão processados pelas tarefas (*tasks*) *Map*, ordenados na fase *Shuffle* e inseridos nas tarefas *Reduce* de forma paralela. Os arquivos de entrada e saída são armazenados no sistema de arquivos e o sistema é responsável pelo escalonamento, agendamento e reexecução de tarefas que tenham falhado (MAPREDUCE. . . , 2022).

A estrutura Cliente/Servidor tem como nodo Master o *JobTracker* e como nodos Worker os *TaskTrackers*. Como apresentado anteriormente, o nodo Master designa as tarefas e os nodos Worker as executam. A aplicação do programador fornece o local dos arquivos de entrada e saída, a implementação das funções de *Map* e *Reduce* e outros parâmetros de configuração do *Job*. Então, o cliente *Hadoop* envia o *Job* e o arquivo de configuração para o *JobTracker* que distribui as tarefas e controla o funcionamento desse *Job*.

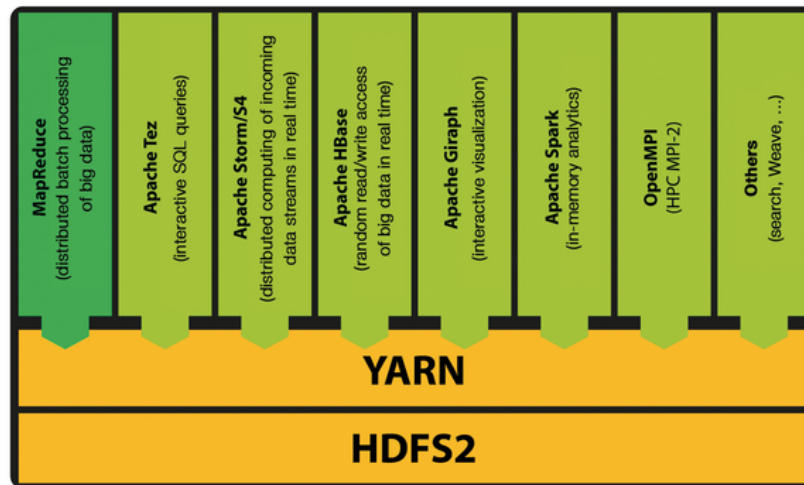
### 2.3.4 Hadoop YARN

O *Hadoop YARN* é um subprojeto que tem como objetivo dividir as funcionalidades de gerenciamento de recursos de escalonamento de tarefas em módulos diferentes, tendo então um gerenciador global de recursos e um gerenciador local por aplicação (GOLDMAN et al., 2012).

O gerenciador global trabalha em conjunto com um gerenciador de nodos responsável pelos contêineres e pelo monitoramento de uso de recursos, como CPU, memória, uso de disco e uso de redes, assim como o repasse dessas informações para o gerenciador global (APACHE. . . , 2022).

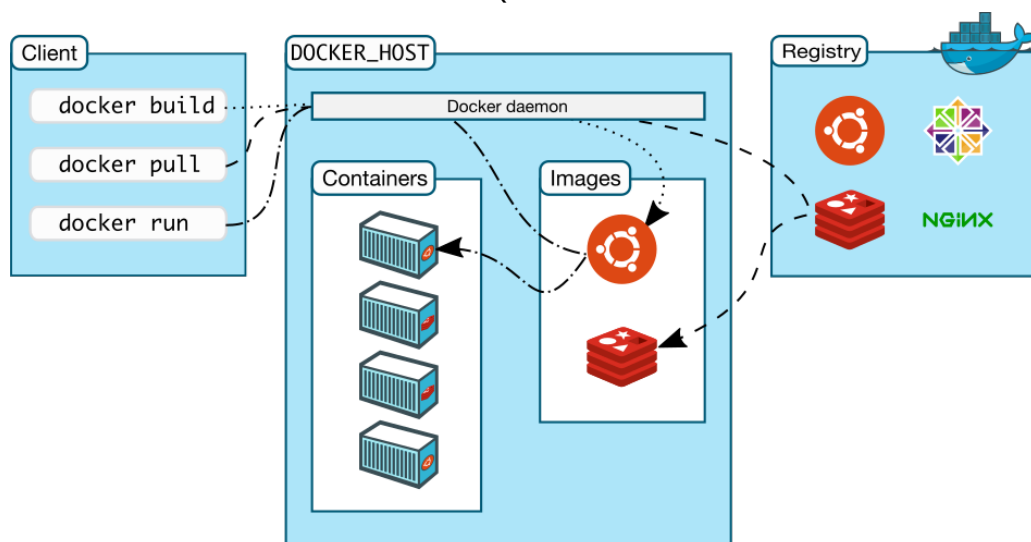
O YARN foi adicionado ao *Hadoop* versão 2.0 permitindo a separação das camadas de gerenciamento de recursos que possam ser alocados pela aplicação. Com essa camada independente, ilustrada na FIGURA 3, as aplicações *MapReduce* podem ser utilizadas em conjunto com aplicações não *MapReduce*. Além disso, esse formato de implementação possibilita economizar custos com o melhor aproveitamento dos nodos (KOBYLINSKA; MARTINS, 2014).

FIGURA 3 – NOVA ARQUITETURA DO HADOOP 2.0



FONTE: (KOBYLINSKA; MARTINS, 2014)

FIGURA 4 – ARQUITETURA DOCKER



FONTE: (DOCKER..., 2022)

## 2.4 VIRTUALIZAÇÃO

Virtualização é o processo de criar um ambiente ou uma versão virtual de algum componente computacional, tal como *hardwares*, dispositivos de armazenamento e recursos de rede. A virtualização permite que haja economia nos custos de *hardware*, melhoria na recuperação em caso de falhas e redução da necessidade de espaço físico para *datacenters* (PORTNOY, 2012).

Uma das técnicas da virtualização é a utilização de contêiners. Contêiners, uma virtualização a nível de sistema, permitem que existam múltiplos espaços do usuário por cima de um determinada kernel de sistema.

Docker é uma ferramenta que tem como objetivo automatizar a implantação de aplicações em contêineres, cuja estrutura geral está ilustrada na FIGURA 4. Essa ferramenta empilha uma implantação de uma aplicação em cima de um ambiente de execução em um contêiner, ou seja, simula um ambiente virtual de modo que o programador possa trabalhar com sua aplicação em produção de forma extremamente configurável para as suas necessidades. Para isso, o Docker utiliza um recurso de imagem, que se refere aos arquivos de sistemas que determinada aplicação necessita para ser executada. Então, esses arquivos são empilhados entre si e servem como uma receita para construção de um ou de múltiplos contêineres (TURNBULL, 2014).

### 3 OTIMIZAÇÃO DO MAPREDUCE

Como visto anteriormente, o *MapReduce* é um processo com diversas etapas, e portanto, há muitas permutações possíveis das suas configurações que permitem a melhora da performance que, através da modificação dos parâmetros de configuração, é chamada de *tuning*. Para White (2015), os fatores do *Job* mais relevantes a serem considerados com objetivo de obter aumento da performance são exibidos no QUADRO 1.

QUADRO 1 – FATORES PARA *TUNING* DO *JOB MAPREDUCE*

| ÁREA A SER OTIMIZADA            | COMO OTIMIZAR  |
|---------------------------------|--|
| Quantidade de mapeadores        | Verificar se é possível diminuir a quantidade de execuções da função <i>Map</i> de modo que cada uma seja executada por mais tempo. O tempo médio recomendado na literatura é de 1 minuto. |
| Quantidade de redutores         | Verificar se mais de um redutor está sendo utilizado. O recomendado é que cada tarefa seja executada em média durante 5 minutos e produza 1 bloco de dados.                                |
| Uso de combinadores             | Verificar se é possível utilizar algum combinador de dados de modo que a quantidade de data passada à função <i>Shuffle</i> seja menor.  |
| Compressão                      | Usualmente, o tempo de execução de um <i>Job</i> é diminuído ao usar compressão de dados.  |
| Ajustes na parte <i>Shuffle</i> | A parte <i>Shuffle</i> do processo possui vários parâmetros de <i>tuning</i> de memória que podem ser utilizados para a melhora da performance do <i>Job</i> .                             |

FONTE: Adaptado de (WHITE, 2015)

#### 3.1 PARÂMETROS DO MAPREDUCE

O *tuning* pode ser realizado através da avaliação e mudança dos parâmetros de configuração do *MapReduce*. Cada parâmetro tem um objetivo específico e pode melhorar uma característica do processo. Algumas variáveis mudam configurações no *Job* e algumas afetam o *cluster* diretamente.

*Hadoop* foi criado para processar grandes arquivos de entradas e é otimizado para *clusters* em máquinas heterogêneas, ou seja, sistemas que usam mais de um tipo de processador com o objetivo de melhorar a performance. Cada *Job* segue a seguinte sequência de passos: configuração, fase *shuffle/sort* e fase *reduce*. O *Hadoop* é responsável por configurar e gerenciar cada um desses passos (VENNER, 2009).

O objetivo principal a ser atingido durante o *tuning* é possibilitar que a fase *Shuffle* tenha a maior quantidade de memória disponível, ao mesmo tempo que as fases *Map* e *Reduce* tenham memória suficiente para funcionar propriamente. A quantidade de memória disponibilizada a cada *Java Virtual Machine* é determinada pelos parâmetros *mapreduce.map.memory.mb* e *mapreduce.reduce.memory.mb* (WHITE, 2015).

A fase *Map* recebe de entrada um arquivo e o parâmetro *dfs.blocksize* é responsável por determinar o tamanho do bloco em *bytes* sobre o qual este arquivo será dividido enquanto o parâmetro *dfs.replication* é responsável por determinar quantos blocos serão criados. Ainda nessa fase, os pares (CHAVE, VALOR) são particionados, e essas partições são ordenadas na fase *Shuffle*. O arquivo criado para cada partição é chamado de *spill*. Para cada tarefa *Reduce* existe um *spill*, que passará por uma ordenação do tipo *Merge Sort*, na segunda etapa da fase *Shuffle*, chamada de *Sort* (VENNER, 2009).

Segundo White (2015), a melhor performance da fase *Map* pode ser obtida através da minimização da quantidade de *spills*, cujos parâmetros de controles são *mapreduce.task.io.sort.factor* - número máximo de entradas para a função *Merge Sort* - e *mapreduce.task.io.sort.mb* - tamanho do *buffer* de memória para a saída da função *Map*. O último é especialmente importante e deve ser aumentado sempre que possível. Quando o *buffer* atinge a capacidade percentual determinada pelo parâmetro (*mapreduce.map.sort.spill.percent*) ocorre um vazamento de memória e os conteúdos restantes do arquivo de saída são colocados no disco (no arquivo chamado de *spill*).

Caso exista mais de uma determinada quantidade de arquivos *spill* (quantidade determinada pela propriedade *mapreduce.map.combine.minspills*), a função combinadora é executada novamente antes de ser criado o arquivo de saída. Uma outra otimização possível é o uso de compressores de dados nos arquivos de saída da fase *Map*, processo facilmente habilitado através dos parâmetros *mapreduce.map.output.compress* - valor booleano que habilita a compressão - e *mapreduce.map.output.compress.codec* - classe que vai realizar a compressão (WHITE, 2015).

A fase *Reduce* é otimizada quando os dados intermediários são armazenados na memória, o que não acontece por padrão, visto que sem a alteração dos parâmetros toda a memória é alocada para a fase *Reduce* em si. As propriedades que podem ser alteradas para atingir esse objetivo são *mapreduce.reduce.merge.inmem.threshold*, *mapreduce.reduce.input.buffer.percent* e *mapreduce.reduce.shuffle.merge.percent*, cujos valores ótimos são 0 e 1.0, respectivamente (WHITE, 2015).

Um dos parâmetros que pode ser otimizado na fase *Reduce* é o *mapreduce.reduce.shuffle.parallelcopies*, que determina a quantidade de tarefas que serão executadas em paralelo pelos *reducers* para copiar os arquivos de saída das tarefas *Map* quando estas são finalizadas e seu valor ótimo depende da quantidade de dados que já passou pela fase *Shuffle* (LI et al., 2014).

Na parte do processo onde as cópias dos arquivos de saída da fase *Map* são feitas, o tamanho do *buffer* é controlado pela propriedade *mapreduce.reduce.shuffle.input.buffer.percent*, que especifica a proporção do *heap* que será usada para a finalidade mencionada. Quando esse *buffer* atinge um número determinado por *mapreduce.reduce.shuffle.merge.percent* ou *mapreduce.reduce.merge.inmem.threshold*, o restante dos arquivos é alocado no disco (WHITE,

2015).

Para cada gerenciador de nodo, o número de partições do arquivo de saída disponibilizados para a fase *Reduce* é determinado pela propriedade *mapreduce.shuffle.max.threads*. O valor padrão de 0 determina que o número máximo de tarefas é o dobro do número de processadores da máquina (VENNER, 2009).

Ainda, as propriedades *mapreduce.output.fileoutputformat.compress*, *mapreduce.output.fileoutputformat.compress* e *mapreduce.output.fileoutputformat.compress* determinam, respectivamente, se os arquivos de saída do *Job* serão comprimidos, qual será a classe responsável pela compressão e como essa compressão ocorrerá. Por fim, o parâmetro *io.file.buffer.size* determina o tamanho do *buffer* que será usado nas operações de leitura e escrita.

Os quadros a seguir resumem os parâmetros mencionados previamente, assim como outros parâmetros relevantes e os valores padrões de cada um:

QUADRO 2 – PARÂMETROS DE AJUSTE DA QUANTIDADE DE TAREFAS *MAP*

| PARÂMETRO                                  | DESCRIÇÃO   | VALOR PADRÃO |
|--|---|--------------|
| <b>mapreduce.task.io.sort.mb</b>           | Tamanho em <i>bytes</i> do <i>buffer</i> de memória na ordenação na saída da função <i>Map</i> .                                | 100          |
| <b>mapreduce.task.io.sort.factor</b>       | Número máximo de arquivos para juntar simultaneamente durante a ordenação.  | 10           |
| <b>mapreduce.map.sort.spill.percent</b>    | Limite de uso do <i>buffer</i> de memória que pode ser usado antes que os dados sejam colocados em disco.                       | 0.80         |
| <b>mapreduce.map.combine.minspills</b>     | Número mínimo de arquivos de vazamento necessário para o combinador funcionar (se um combinador for especificado).              | 3            |
| <b>mapreduce.map.output.compress</b>       | Define se a saída da função <i>Map</i> será comprimida.   | false        |
| <b>mapreduce.map.output.compress.codec</b> | Codificador usado na compressão.  | DefaultCodec |
| <b>mapreduce.shuffle.max.threads</b>       | Número de tarefas <i>Worker</i> por gerenciador de nodos. Esse parâmetro não funciona por <i>Job</i> e sim por <i>cluster</i> . | 0            |

FONTE: Adaptado de (APACHE. . . , 2019)



QUADRO 3 – PARÂMETROS DE AJUSTE DA QUANTIDADE DE TAREFAS *REDUCE*

| PARÂMETRO  | DESCRIÇÃO   | VALOR PADRÃO |
|--|---|--------------|
| <b>mapreduce.task.io.sort.factor</b>                 | Número máximo de arquivos para juntar simultaneamente durante a ordenação.  | 10           |
| <b>mapreduce.reduce.shuffle.parallelcopies</b>       | Número de tarefas usadas para copiar saída de funções <i>Map</i> para funções <i>Reduce</i> .   | 5            |
| <b>mapreduce.reduce.shuffle.maxfetchfailures</b>     | Número de vezes que uma tarefa <i>Reduce</i> tenta obter arquivo de entrada.  | 10           |
| <b>mapreduce.reduce.shuffle.input.buffer.percent</b> | Porcentagem de tamanho do <i>heap</i> a ser alocada para a saída da fase <i>Map</i> .   | 0.70         |
| <b>mapreduce.reduce.shuffle.merge.percent</b>        | Limite porcentual da saída da fase <i>Map</i> para iniciar o processo de juntar saídas.   | 0.66         |
| <b>mapreduce.reduce.merge.inmem.threshold</b>        | Quantidade de saídas da função <i>Map</i> para a saída da fase <i>Map</i> .   | 1000         |
| <b>mapreduce.reduce.input.buffer.percent</b>         | Percentual que determinar o tamanho do <i>heap</i> que será utilizado para armazenar saídas da função <i>Map</i> durante a fase <i>Reduce</i> . | 0.0          |
| <b>mapreduce.job.reduce.slowstart.completedmaps</b>  | Percentual de tarefas <i>Map</i> que devem estar completas antes que as tarefas <i>Reduce</i> sejam iniciadas.                                  | 0.05         |

FONTE: Adaptado de (APACHE. . . , 2019)

QUADRO 4 – PARÂMETROS ADICIONAIS DE CONFIGURAÇÃO

| PARÂMETRO   | DESCRIÇÃO  | VALOR PADRÃO   |
|---|--|----------------|
| <b>dfs.blocksize</b>                                    | Tamanho do bloco em <i>bytes</i> sobre o qual arquivo de entrada do <i>Map</i> será. | 67108864 bytes |
| <b>dfs.replication</b>                                  | Quantidade de replicação dos blocos.   | 3              |
| <b>mapreduce.map.memory.mb</b>                          | Memória alocada para cada tarefa <i>Map</i> .  | 1024           |
| <b>mapreduce.reduce.memory.mb</b>                       | Memória alocada para cada tarefa <i>Reduce</i> .                                     | 1024           |
| <b>mapreduce.output.fileoutputformat.compress</b>       | Define se a saída do <i>Job</i> será comprimida.                                     | false          |
| <b>mapreduce.output.fileoutputformat.compress.codec</b> | Codificador usado na compressão.   | DefaultCodec   |
| <b>mapreduce.output.fileoutputformat.compress.type</b>  | Define como os arquivos de saída do <i>Job</i> serão comprimidos.                    | RECORD         |
| <b>io.file.buffer.size</b>                              | Tamanho do <i>buffer</i> que será usado durante operações de leitura e escrita.      | 4096           |

FONTE: Adaptado de (APACHE. . . , 2019)

### 3.2 TERASORT

A técnica de *benchmarking* consiste na execução, por diversas vezes, de um programa (no caso do *MapReduce*, de um *Job*), a fim de testar se os resultados obtidos são os esperados. Esse processo é eficiente porque é possível comparar os diversos resultados obtidos e obter avaliações de performance (WHITE, 2015).

O *Hadoop* possui várias métricas de *benchmarks* embutidas que podem ser utilizadas para melhorar o funcionamento do *Job*, cada uma delas com o objetivo de monitorar um fator diferente, como por exemplo, *TestDFSIO* - responsável por testar a performance dos dispositivos de entrada e saída - e *MRBench/NNBench* - que testam em conjunto vários *Jobs* pequenos múltiplas vezes (WHITE, 2015).

Neste trabalho será utilizado a ferramenta *TeraSort*. Para White (2015), ela é extremamente eficaz no *benchmarking* do *HDFS* e *MapReduce* em conjunto, já que executa e avalia todos os passos do paradigma *MapReduce*. Essa ferramenta funciona em três etapas:

- *TeraGen* executa um *Job* só de funções *Map* que cria um conjunto de dados binários aleatórios. A execução desse comando é exemplificada no CÓDIGO 3.1.

---

```
1  hadoop jar hadoop-mapreduce-examples-*.jar \  
2  teragen <numero de linhas de 100 bytes cada> <diretorio de saida>
```

---

CÓDIGO 3.1 – Exemplo de execução do *TeraGen* adaptado de (WHITE, 2015)

- *TeraSort* executa a ordenação dos dados. É nesse passo que é avaliada a performance do *MapReduce*, pois é aqui que as operações do paradigma são executadas. A execução desse comando é exemplificada no CÓDIGO 3.2.

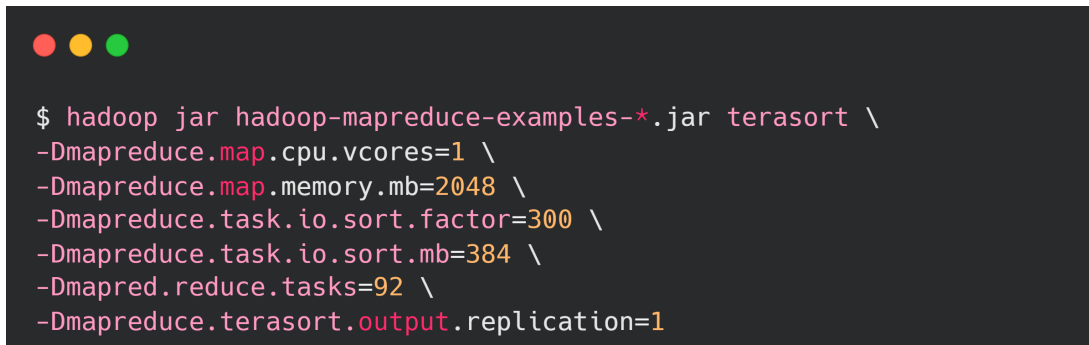
---

```
1  hadoop jar hadoop-mapreduce-examples-*.jar \  
2  terasort <diretorio de entrada> <diretorio de saida>
```

---

CÓDIGO 3.2 – Exemplo de execução do *TeraSort* adaptado de (WHITE, 2015)

- *TeraValidate* executa checagens nos dados ordenados resultantes da fase anterior para verificar se a ordenação foi feita corretamente. A execução desse comando é exemplificada no CÓDIGO 3.3.

FIGURA 5 – EXECUÇÃO DO *TERASORT* COM PARÂMETROS DE *TUNING*


```
$ hadoop jar hadoop-mapreduce-examples-*.jar terasort \
-Dmapreduce.map.cpu.vcores=1 \
-Dmapreduce.map.memory.mb=2048 \
-Dmapreduce.task.io.sort.factor=300 \
-Dmapreduce.task.io.sort.mb=384 \
-Dmapred.reduce.tasks=92 \
-Dmapreduce.terasort.output.replication=1
```

FONTE: A autora (2022)

---

```
1  hadoop jar hadoop-mapreduce-examples-*.jar \
2  teravalidate <arquivo de entrada (diretorio de saida do
   terasort)> <diretorio de saida>
```

---

CÓDIGO 3.3 – Exemplo de execução do *TeraValidate* adaptado de (WHITE, 2015)

Os parâmetros de configuração do *tuning* podem ser usados no comando que executa o *TeraSort*, processo exemplificado na FIGURA 5.

### 3.3 AMBIENTE EXPERIMENTAL

O ambiente no qual serão realizadas as execuções e testes tem as seguintes configurações:

- MacBook Pro com processador Intel Core i7 6-core, 2,6 GHz de velocidade de processamento e cache compartilhada L3 de 12MB
- 16GB de memória RAM DDR4
- SSD PCIe de 512GB
- Configurações gráficas: AMD Radeon Pro 5300M 4 GB e Intel UHD Graphics 630 1536 MB
- Docker versão 20.10.11
- Hadoop versão 3.2.1

Usando a imagem do *Hadoop* para Docker disponibilizada pelo Big Data Europe (BIGDATAEUROPE, 2020), foi configurado um *cluster Hadoop* em um contêiner Docker com 3 *datanodes (workers)*, um *namenode HDFS (master)*, um gerenciador de recursos YARN, um servidor com histórico de operações e um gerenciador de nodos.

### 3.4 RESULTADOS

Para que o objetivo de melhora de performance através do *tuning* dos parâmetros fosse atingido, foram feitos testes repetidos utilizando o *Terasort* com variação dos valores padrões de cada uma das propriedades relevantes mencionadas previamente de acordo com sugestões de White (2015) e Venner (2009), assim como a análise de cada uma das execuções da ferramenta de *benchmarking* para que fossem decididas quais mudanças teriam mais impacto no resultado. As execuções do *TeraSort* foram realizadas após a geração de dados aleatórios de entrada pelo *TeraGen* de um arquivo de 10GB.

Os primeiros ajustes realizados foram nas propriedades relacionadas à quantidade de memória disponível, isto é, na memória disponibilizada aos *buffers* utilizados nas operações de leitura e escrita e na saída da função *Map*, assim como às funções *Map* e *Reduce*, o que se provou de grande importância porque diminui consideravelmente o tempo de execução do programa. Isso ocorre devido ao fato de que, ao disponibilizar mais memória aos programas, menos dados são copiados para o disco, economizando tempo.

Depois, foram feitas alterações nas propriedades referentes a compressão de arquivos, habilitando-se a compressão na saída da fase *Map* e do *Job* utilizando a classe *org.apache.hadoop.io.compress.Lz4Codec* para realizar essa operação. A compressão é relevante porque permite que os recursos computacionais não fiquem parados esperando operações de entrada e saída em disco finalizarem por estarem trabalhando com arquivos muito grandes.

Os parâmetros relativos aos blocos sobre os quais os arquivos de entrada da função *Map* são separados também foram modificados, tendo seu tamanho aumentado e sua taxa de replicação diminuída. Esses dois fatores permitem que os blocos sejam de maior tamanho, mas não sejam duplicados em nodos diferentes, resultando em menos operações de leitura e escrita e menos uso de rede.

Outras propriedades da tarefa *Map* que foram alteradas são as que determinam a quantidade de arquivos para juntar simultaneamente e a da taxa limite do *buffer* para que os valores deste sejam transferidos para o disco.

Em relação à fase *Shuffle*, o único parâmetro modificado foi o *mapreduce.shuffle.max.threads*, determinando-se que apenas uma tarefa *Worker* fosse usada pelo gerenciador de nodos.

Por fim, na fase *Reduce*, as propriedades que tiveram efeito relevante na performance foram as que determinam a quantidade de transferências paralelas executadas pela função *Reduce* durante a fase de cópia dos arquivos da fase *Shuffle*, o limite de arquivos para o processo de junção antes de acontecer transferência para o disco, a quantidade de memória a ser alocada para armazenar arquivos de saída da fase *Map* durante a fase *Shuffle* e a porcentagem de tarefas *Map* que deveriam estar finalizar antes de ser iniciado o processo de *Reduce*.

Os parâmetros e seus valores que foram alterados por afetarem a performance estão descritos no QUADRO 5.

QUADRO 5 – PARÂMETROS AJUSTADOS DURANTE O *TUNING*

| PARÂMETRO  | VALOR FINAL |
|--|-------------|
| mapreduce.map.output.compress                    | true        |
| mapreduce.map.output.compress.codec              | Lz4Codec    |
| dfs.blocksize                                    | 335544320   |
| dfs.replication                                  | 1           |
| mapreduce.output.fileoutputformat.compress       | true        |
| mapreduce.output.fileoutputformat.compress.codec | Lz4Codec    |
| mapreduce.output.fileoutputformat.compress.type  | BLOCK       |
| mapreduce.map.memory.mb                          | 2048        |
| io.file.buffer.size                              | 131072      |
| mapreduce.task.io.sort.mb                        | Lz4Codec    |
| mapreduce.task.io.sort.factor                    | 256         |
| mapreduce.map.sort.spill.percent                 | 400         |
| mapreduce.shuffle.max.threads                    | 1.0         |
| mapreduce.reduce.shuffle.parallelcopies          | 20          |
| mapreduce.reduce.merge.inmem.threshold           | 2000        |
| mapreduce.reduce.input.buffer.percent            | 0.8         |
| mapreduce.job.reduce.slowstart.completedmaps     | 0.7         |

FONTE: A autora(2022)

Alguns parâmetros mencionados anteriormente - resumidos no QUADRO 6 não tiveram efeito na performance no programa, seja pelo tamanho do arquivo de dados inicial pelas configurações do ambiente experimental. Por causa disso, seus valores padrões foram mantidos nas execuções.

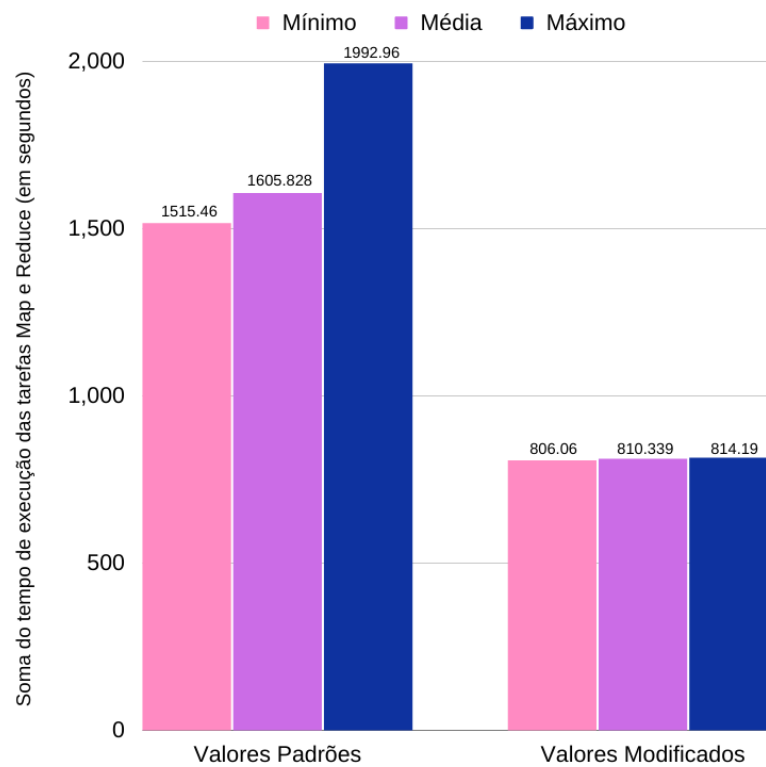
QUADRO 6 – PARÂMETROS NÃO AJUSTADOS DURANTE O *TUNING*

| PARÂMETRO                                     | VALOR FINAL |
|---|-------------|
| mapreduce.map.combine.minspills               | 3           |
| mapreduce.reduce.shuffle.maxfetchfailures     | 10          |
| mapreduce.reduce.shuffle.input.buffer.percent | 0.7         |
| mapreduce.reduce.shuffle.merge.percent        | 0.66        |

FONTE: A autora(2022)

Os resultados relevantes da aplicação do *tuning* são os tempos obtidos na execução do *TeraSort*, os quais podem ser obtidos pela sua saída que mostra, em milissegundos, o tempo utilizado pelas fases *Map* e *Reduce* do processo. Segundo Fleming e Wallace (1986), no caso de *benchmarking* de métricas de tempo, é apropriado usar uma média aritmética padrão para avaliar os resultados. Além disso, apresentar os valores mínimos e máximos obtidos na execuções de modo que uma visão geral da performance possa ser exemplificada.

Para geração dessas métricas, o *TeraSort* foi executado 10 vezes com os parâmetros com valores padrão e 10 vezes com os parâmetros com valores alterados.

FIGURA 6 – RESULTADOS DO *TUNING*

FONTE: A autora (2022)

Dessa forma, a FIGURA 6 acima ilustra os resultados obtidos nesse experimentos antes e depois do processo de aplicação do *tuning*. Como é possível ver, a mudança dos parâmetros teve um grande impacto da performance do *Hadoop MapReduce*, diminuindo a execução das suas principais funções em quase 50%.

## 4 CONCLUSÃO

Nesse trabalho foram apresentados o paradigma *MapReduce*, assim como o *framework Hadoop* e a junção destes, o *Hadoop MapReduce*, assim como o detalhamento do seu funcionamento, sua estrutura e, principalmente, os parâmetros de configuração que se mostraram extremamente influentes na sua boa performance.

A partir do estudo dessas ferramentas assim como testes práticos executados, ficou claro como o bom conhecimento do problema a ser resolvido pela aplicação, a ferramenta que está sendo utilizada e o sistema sobre o qual essa ferramenta está sendo executada são relevantes e devem ser consideradas para a configuração do *Hadoop MapReduce* de forma a obter a melhor performance possível.

Os experimentos realizados foram aplicados com uma quantidade pequena de dados - em relação a quantidades observadas no mundo real - mas mesmo assim foi possível otimizar os resultados dos *benchmarks*.

Em relação a trabalhos futuros sobre o tema, é interessante ressaltar que já existem em progresso estudos e artigos sobre ferramentas que realizam o *tuning* automático do *Hadoop MapReduce*, inclusive com aprendizagem de máquina e alterações em tempo real. Com isso, temas interessante a serem considerados incluem a análise dessas ferramentas, a comparação entre elas da qual obtém o melhor resultado ou até mesmo a implementação de uma nova ferramenta com técnicas ainda não utilizadas.

## REFERÊNCIAS

AMRAM, M.; STATSKY, J. **Pandemonium**. v. 3. [S.l.: s.n.], jan. 2019. Citado 1 vez na página 4.

APACHE Hadoop 3.2.1. [S.l.]: Apache Software Foundation, set. 2019. Disponível em: <https://hadoop.apache.org/docs/r3.2.1/>. Citado 0 vezes nas páginas 23, 24.

APACHE Hadoop YARN. [S.l.: s.n.], 2022. Disponível em: <https://hadoop.apache.org/docs/stable/hadoop-yarn/hadoop-yarn-site/YARN.html>. Citado 1 vez na página 18.

BELCASTRO, L.; CANTINI, R.; MAROZZO, F.; ORSINO, A.; TALIA, D.; TRUNFIO, P. Programming big data analysis: principles and solutions. **Journal of Big Data**, SpringerOpen, v. 9, n. 1, p. 1–50, 2022. Citado 2 vez na página 11.

BIGDATAEUROPE. **docker-hadoop**. [S.l.]: GitHub, 2020. Disponível em: <https://github.com/big-data-europe/docker-hadoop>. Citado 1 vez na página 26.

DEAN, J.; GHEMAWAT, S. MapReduce: simplified data processing on large clusters. **Communications of the ACM**, ACM New York, NY, USA, v. 51, n. 1, p. 107–113, 2008. Citado 12 vezes nas páginas 7, 11, 13–16.

DOCKER Overview. [S.l.]: Docker Inc., abr. 2022. Disponível em: <https://docs.docker.com/get-started/overview/>. Citado 0 vez na página 19.

FLEMING, P. J.; WALLACE, J. J. How not to lie with statistics: the correct way to summarize benchmark results. **Communications of the ACM**, ACM New York, NY, USA, v. 29, n. 3, p. 218–221, 1986. Citado 1 vez na página 28.

GOLDMAN, A.; KON, F.; JUNIOR, F. P.; POLATO, I.; FÁTIMA PEREIRA, R. de. Apache Hadoop: conceitos teóricos e práticos, evolução e novas possibilidades. **XXXI Jornadas de atualizações em informatica**, p. 88–136, 2012. Citado 3 vezes nas páginas 13, 17, 18.

HDFS architecture guide. [S.l.: s.n.], out. 2020. Disponível em: <https://hadoop.apache.org/docs/r3.2.1/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>. Citado 4 vez na página 17.



KHALEEL, A.; AL-RAWESHIDY, H. Optimization of computing and networking resources of a Hadoop cluster based on software defined network. **IEEE Access**, IEEE, v. 6, p. 61351–61365, 2018. Citado 1 vez na página 11.

KOBYLINSKA, A.; MARTINS, F. **Big data tools for midcaps and others**. [S.l.: s.n.], 2014. Disponível em:  
<https://www.admin-magazine.com/Archive/2014/20/Big-data-tools-for-midcaps-and-others>.  
 Citado 1 vezes nas páginas 18, 19.

KUO, Y.-H.; KUSIAK, A. From data to big data in production research: the past and future trends. **International Journal of Production Research**, Taylor & Francis, v. 57, n. 15-16, p. 4828–4853, 2019. DOI: [10.1080/00207543.2018.1443230](https://doi.org/10.1080/00207543.2018.1443230). eprint:  
<https://doi.org/10.1080/00207543.2018.1443230>. Disponível em:  
<https://doi.org/10.1080/00207543.2018.1443230>. Citado 1 vez na página 11.

LI, M.; ZENG, L.; MENG, S.; TAN, J.; ZHANG, L.; BUTT, A. R.; FULLER, N. Mronline: Mapreduce online performance tuning. In: PROCEEDINGS of the 23rd international symposium on High-performance parallel and distributed computing. [S.l.: s.n.], 2014. P. 165–176. Citado 1 vez na página 22.

MAPREDUCE Tutorial. [S.l.: s.n.], 2022. Disponível em:  
[https://hadoop.apache.org/docs/r1.2.1/mapred\\_tutorial.html](https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html). Citado 1 vez na página 18.

NAVARRO BELMONTE, V. P. **Improving Real Time Tuning on YARN**. 2018. Tese (Doutorado) – Carleton University. Citado 1 vez na página 16.

PORTNOY, M. **Virtualization essentials**. [S.l.]: John Wiley & Sons, 2012. v. 19. Citado 1 vez na página 19.

RYDNING, D. R.-J. G.-J. The digitization of the world from edge to core. **Framingham: International Data Corporation**, p. 16, 2018. Citado 1 vez na página 11.

SADASHIV, N.; KUMAR, S. D. Cluster, grid and cloud computing: A detailed comparison. In: IEEE. 2011 6th international conference on computer science & education (ICCSE). [S.l.: s.n.], 2011. P. 477–482. Citado 2 vez na página 13.

TURNBULL, J. **The Docker Book: Containerization is the new virtualization**. [S.l.]: James Turnbull, 2014. Citado 1 vez na página 20.

VENNER, J. **Pro hadoop**. [S.l.]: Apress, 2009. Citado 5 vezes nas páginas 14, 21–23, 27.

WHITE, T. **Hadoop: The definitive guide**. [S.l.]: "O'Reilly Media, Inc.", 2015. Citado 21 vezes nas páginas 7, 11, 14, 21, 22, 25–27.

## GLOSSÁRIO

**BENCHMARK** Execução de um programa várias vezes com objetivo de avaliar sua performance.

**BUFFER** Região de memória computacional usada para armazenar dados temporariamente.

**BYTE** Unidade de dados que contém 8 bits.

**DATA WAREHOUSE** Sistema de gerenciamento de grandes quantidades de dados usado principalmente na área de Business Intelligence.

**DATA CENTERS** Espaço físico dedicado que reúne sistemas de computadores ou de armazenamento.

**FRAMEWORK** Abstração de na qual um *software* provê uma funcionalidade genérica que pode ser incrementada com código do usuário.

**HARDWARE** Todo componente físico de um computador.

**HEAP** Estrutura de dados baseada em árvores.

**JAVA VIRTUAL MACHINE** Máquina virtual que executa programas na linguagem Java ou em outra linguagem de programação caso estejam compilados como código binário Java.

**MERGE SORT** Eficiente algoritmo de ordenação baseado em comparação de valores.

**TUNING** Melhora da performance de um sistema através da mudança de partes do sistema que mais influenciam sua execução.