**Pontifícia Universidade Católica de Minas Gerais**

Programa de Pós-Graduação em Informática

Edgar Paes Cassemiro

# ANÁLISE DO IMPACTO DOS AJUSTES DE PARÂMETROS DE CONFIGURAÇÕES NOS SUBSISTEMAS DO SISTEMA OPERACIONAL LINUX UTILIZANDO A APLICAÇÃO APACHE HADOOP

Belo Horizonte

2017

Edgar Paes Cassemiro

# ANÁLISE DO IMPACTO DOS AJUSTES DE PARÂMETROS DE CONFIGURAÇÕES NOS SUBSISTEMAS DO SISTEMA OPERACIONAL LINUX UTILIZANDO A APLICAÇÃO APACHE HADOOP

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Humberto Torres Marques Neto

Belo Horizonte

2017

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Cassemiro, Edgar Paes
C344a Análise do impacto dos ajustes de parâmetros de configurações nos subsistemas do sistema operacional Linux utilizando a aplicação Apache Hadoop / Edgar Paes Cassemiro. Belo Horizonte, 2017.
111 f. : il.

Orientador: Humberto Torres Marques Neto
Dissertação (Mestrado) - Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática

1. Apache (Programa de computador). 2. Big data. 3. Linux (Sistema operacional de computador). 4. Desempenho. 5. Sistemas operacionais (Computadores). I. Marques Neto, Humberto Torres. II. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática. III. Título.



CDU: 681.3.03
Ficha catalográfica elaborada por Rosane Alves Martins da Silva – CRB 6/2971

Edgar Paes Cassemiro

# ANÁLISE DO IMPACTO DOS AJUSTES DE PARÂMETROS DE CONFIGURAÇÕES NOS SUBSISTEMAS DO SISTEMA OPERACIONAL LINUX UTILIZANDO A APLICAÇÃO APACHE HADOOP

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial à obtenção do título de Mestre em Informática.

---

Prof. Dr. Humberto Torres Marques Neto (Orientador)
Pontifícia Unversidade Católica de Minas Gerais (PUC Minas)

---

Prof. Dr. Henrique Cota de Freitas
Pontifícia Unversidade Católica de Minas Gerais (PUC Minas)

---

Prof. Dr. Dorgival Olavo Guedes Neto
Universidade Federal de Minas Gerais (UFMG)

Belo Horizonte, 27 de Outubro de 2017.

## DEDICATÓRIA

Para minha mãe.

## AGRADECIMENTO

A Deus, sempre.

Familiares e esposa.

Ao Jornal OTempo (Grupo SADA) por fornecer toda a infraestrutura necessária para realização dos experimentos e disponibilização de horas de estudo e pesquisa.

## RESUMO

O Apache Hadoop é uma das aplicações de código aberto mais utilizadas para tratamento massivo de dados. O Linux é o sistema operacional que majoritariamente oferece o embarco tecnológico para tal aplicação. Vários trabalhos propõem realizar ajustes de desempenho tanto nas estruturas internas do sistema de arquivos HDFS quanto nos parâmetros principais de configurações oferecidos pelo Apache Hadoop. Sendo tais pesquisas muito bem consolidadas, este trabalho propôs analisar outra frente: ajustes de desempenho especificamente no sistema operacional, no caso, o Linux.

Desta forma, este trabalho realizou um estudo detalhado dos parâmetros de configurações dos principais subsistemas do sistema operacional Linux, os quais compreendem, o subsistema de tarefas, memória virtual, sistema de arquivos e I/O de disco, que quando ajustados adequadamente permitem obter melhor desempenho em aplicações de missão crítica dadas as suas características de *CPU-bound* e *I/O-bound* inerentes.

Neste trabalho conseguimos resultados com melhor desempenho modificando os parâmetros de configurações do sistema operacional Linux. Os parâmetros de configurações com seus respectivos ajustes foram catalogados com os valores que produziram melhor desempenho.



Palavras-chave: Apache Hadoop. Big Data. Linux. Desempenho.

## ABSTRACT

Apache Hadoop is one of the most widely used open source application for massive data processing. Linux is mostly used operating system to offer the technology base for such an application. Several works propose performing performance adjustments in both the internal structures of the HDFS file system and the main settings parameters offered by Apache Hadoop. Being such researches very well consolidated, this work proposed to analyze another front: performance adjustments specifically in the operating system, in this case, Linux.

Thus, this work carried out a detailed study of the configuration parameters of the main subsystems of the Linux operating system, which comprise the task subsystem, virtual memory, file system and disk I/O, which when properly adjusted, allow better performance in critical-application given their inherent CPU-bound and I/O-bound characteristics.

In this work we have obtained results with better performance by modifying the parameters of Linux operating system configurations. The settings parameters with their respective settings were cataloged with the values that produced better performance.



Keywords: Apache Hadoop. Big Data. Linux. Performance.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE SIGLAS

| | |
|---|---|
| SMP | *Symetric Multiprocessing* |
| UMA | *Uniform Memory Access* |
| NUMA | *Non Uniform Memory Access* |
| CFS | *Completely Fair Scheduler* |
| HDFS | *Hadoop Distributed File System* |
| MMU | *Memory Management Unit* |
| VFS | *Virtual File System* |
| JVM | *Java Virtual Machine* |
| IRQ | *Interrupt Request Queue* |
| RAM | *Random Access Memory* |
| CPU | *Central Processing Unit* |
| OOM | *Out Of Memory* |
| CFQ | *Completely Fair Queuing* |
| NOOP | *No-Operation* |
| CAV | *Constant Angular Velocity* |
| RAID | *Redundant Array of Independent Disks* |
| LVM | *Logical Volume Manager* |
| ZFS | *Zettabyte File System* |
| SAS | *Serial Attached SCSI* |
| SATA | *Serial ATA* |
| SAN | *Storage Area Network* |
| LRU | *Least Recently Used* |
| NRU | *Not Recently Used* |

## SUMÁRIO

# 1 INTRODUÇÃO

Sistemas de missão crítica no âmbito corporativo são responsáveis por suportar tecnologicamente toda a área de negócios e garantir a operacionalização dos processos envolvidos. Uma típica infraestrutura de tecnologia da informação corporativa inclui serviços como correio eletrônico, serviços Web, banco de dados, aplicações *core business* e aplicações ERP (*Enterprise Resource Planning*). O desempenho do sistema de missão crítica é um fator decisivo para a continuidade de negócio.

Além dos serviços típicos de uma infraestrutura de tecnologia da informação corporativa, tecnologias emergentes (como o paradigma da Internet das Coisas (IoT)) estarão cada vez mais alinhadas no âmbito corporativo. Inclui-se também as redes veiculares e cidades inteligentes bem como a fomentação de tecnologias subjacentes, como IPv6, redes 5G, redes de sensores e sistemas embutidos.

Tais tecnologias envolvem o tratamento massivo de dados. Uma solução de software de código aberto muito utilizada e apoiada por grandes empresas como Facebook, Yahoo!, Amazon, LinkedIn, Lastfm, Twitter, entre outras, é o Apache Hadoop. O Apache Hadoop é uma implementação do modelo de programação Map/Reduce e surgiu como um dos sistemas mais populares para processamento massivo de dados e *Big Data* (ANTYPAS; ZACHEILAS; KALOGERAKI, 2015). (LOUGHRAN, 2016) apresenta uma lista com mais de noventa grandes grupos, incluindo empresas, universidades e órgãos governamentais que utilizam o Apache Hadoop.

O Apache Hadoop é uma aplicação *multi-threaded*, assim como diversas outras aplicações típicas de uma infraestrutura de tecnologia da informação corporativa e portanto, possui características de *CPU-bound* e *I/O-bound* semelhantes.

Os termos *CPU-bound* e *I/O-bound* são utilizados no âmbito de sistemas operacionais (e em outros) para contextualizar as tarefas que apresentam, em geral, longos surtos de uso da CPU e esporádicas esperas por I/O (*CPU-bound*) e tarefas que apresentam, em geral, pequenos surtos de uso da CPU e esperas frequentes por I/O (*I/O-bound*) (TANENBAUM, 2010). Neste sentido, o estudo dos aspectos de desempenho do Apache Hadoop favorece de igual modo tais aplicações.

Há trabalhos que tratam de otimizações específicas no Apache Hadoop (CHO et al., 2013), (LI et al., 2014), (XU; LUO; WOODWARD, 2012), mas este trabalho concentrou-se na otimização da base operacional, no caso o Linux, para que o tratamento de dados tenha melhor desempenho quanto aos seus principais subsistemas.

A otimização da base operacional para suportar o tratamento de grande demanda em aplicações de missão crítica é uma área dentro do **planejamento de capacidade** conhecida como **ajustes de desempenho** (ou *performance tuning*).

Os *guidelines* de sites especializados e sites oficiais como o do Apache Hadoop, Red Hat, Debian, Ubuntu e SUSE Linux, oferecem pouco conteúdo abordando ajustes de desempenho, apesar de enfaticamente o fomentarem. Encontram-se poucos experimentos em ambientes de virtualização, são orientados ao produto oferecido, questões particulares de *CPU-bound* e *I/O-bound* da aplicação não são consideradas e majoritariamente se traduzem em conteúdo de referência técnica.

## 1.1 Objetivo Geral

No âmbito do sistema operacional Linux temos as seguintes questões: **(i)** o sistema operacional Linux ao ser instalado em um determinado *hardware* possui parâmetros de configurações ajustados que permitem atender os requisitos de aplicações de missão crítica quanto à *CPU-bound* e *I/O-bound*? **(ii)** tais parâmetros de configurações podem ser ajustados para atender aos requisitos de aplicações específicas, como banco de dados, servidores de arquivo, servidores Web, servidores de aplicação? **(iii)** quais parâmetros de configurações que de fato impactam no desempenho de aplicações de missão crítica?

Na perspectiva destas formulações, considerando suas relações diretas, temos como objetivo geral analisar os subsistemas de tarefas, memória virtual, arquivos e I/O de disco do sistema operacional Linux identificando os principais parâmetros de configurações que quando ajustados fornecem melhor desempenho ao sistema.

Sendo assim, temos a **finalidade**, a qual é analisar os principais parâmetros de configurações do sistema operacional Linux que impactam no desempenho. Também, é aqui constituído o **escopo** deste trabalho; o sistema operacional Linux.

### 1.1.1 *Objetivos Específicos*

Consideramos os seguintes **objetivos específicos** para que a finalidade e o escopo sejam alcançados:

**i.** Analisar os subsistemas de tarefas, memória virtual, sistema de arquivos e I/O de disco do sistema operacional Linux.

**ii.** Analisar os principais parâmetros de configurações do kernel, por meio do sistema de arquivos ProcFS e SysFS (subestrutura /proc/sys) que são passíveis de ajustes em função das características peculiares de *CPU-bound* e *I/O-bound*.

**iii.** Realizar experimentos com as principais ferramentas de caracterização de carga de trabalho (*workload*) nativas do Apache Hadoop (e outras) para comparações elementares a partir da metodologia proposta por este trabalho.

Com esses objetivos definidos conseguimos compreender nuances nos subsistemas do sistema operacional Linux que impactam no desempenho das aplicações. Também, a compreesão dos subsistemas do kernel do Linux permitiram uma precisão para atribuição dos ajustes nos parâmetros identificados. As cargas de trabalho foram fundamentais pelo fato de conseguirmos analisar cada subsistema individualmente, suas relações particulares e o comportamento do sistema quando os ajustes nos parâmetros de configurações foram realizados.

Este trabalho analisou os ajustes de parâmetros de configurações no sistema operacional Linux tendo como aplicação base o Apache Hadoop em ambiente de virtualização. Foram catologados os parâmetros em que obtivemos impacto no desempenho do sistema operacional Linux bem como os valores atribuídos a estes parâmetros que conduziram ao melhor resultado.

Neste trabalho, além do referencial teórico e trabalhos relacionados, estruturamos o texto contendo uma visão geral do sistema operacional Linux e do Apache Hadoop no capítulo 4. Este capítulo é o ponto inicial da compreensão dos aspectos fundamentais que serão abordados neste trabalho quanto aos subsistemas do sistema operacional Linux.

A metodologia utilizada neste trabalho é apresentado na capítulo 5. Nesta seção também abordamos as ferramentas utilizadas durante os experimentos, versões de software e ambiente de exeperimentos.

Os conceitos gerais pertinentes ao subsistema de tarefas e algumas de suas particularidades foram abordadas no capítulo 6. Por meio deste capítulo, descrevemos os principais parâmetros de configurações que após as análises e experimentos consideramos fundamentalmente relevantes no aspecto de desempenho do sistema. Sendo assim, buscamos apresentar como a troca de contexto, latências e migrações de CPU impactam negativamente no desempenho do sistema. Também, apresentamos as soluções propostas e os parâmetros de configurações relacionados.

Os pontos principais do subsistema de memória virtual foram abordados no capítulo 7. Iniciamos o capítulo com os conceitos correlatos ao subsistema de memória virtual e também apresentamos os pontos nos quais existem impactos no desempenho e na escalabilidade das aplicações. Os parâmetros de configurações inerentes ao subsistema de memória virtual são também pontuados.

As definições e visão geral dos agendadores de I/O de disco e do sistema de arquivos Ext4 foram abordadas no capítulo 8. Neste capítulo, também são apresentadas nossas considerações quanto ao sistema de armazenamento subjacente, o tamanho do bloco e considerações relevantes de outros trabalhos que corroboram nossas análises e experimentos realizados.

A análise do resultados obtidos, apresentação das tabelas comparativas, gráficos e outras métricas de análise foram apresentos no capítulo 9. Neste capítulo, apresentamos resultados obtidos a partir de cada subsistema, analisando-os separadamente.

Por fim, temos a conclusão deste trabalho e a perspectiva de trabalhos futuros no capítulo 10. Também, elaboramos uma seção com um apêndice contendo todos os parâmetros analisados durante este trabalho e as respectivas observações pertinentes. Ainda, apresentamos dados contendo o desvio padrão das execuções realizadas durante os ajustes dos parâmetros de configurações dos subsistemas do sistema operacional Linux.

# 2 REFERENCIAL TEÓRICO

O referencial teórico, ou seja, o conjunto de artefatos das teorias que são as bases para orientação da pesquisa (PRODANOV; FREITAS, 2013), foi elaborado a partir de três áreas que permitam abordar de forma abrangente, permeando os conceitos do sistema operacional Linux, as otimizações pertinentes ao sistema operacional Linux e os conceitos e otimizações pertinentes ao Apache Hadoop. Os critérios envolvidos na seleção bibliográfica foram as realizações práticas, análises de tendências e abordagens propositivas de ajustes de desempenho, sejam elas no Apache Hadoop (que esclarecem conceitos de mecanismos internos, como o HDFS) ou no sistema operacional Linux.

Portanto, construímos o referencial teórico elaborado pelas seguintes diretivas:

**i.** Da contextualização geral do Apache Hadoop, em que foram analisados os trabalhos (artigos, livros e documentações oficiais) que tratam de forma panorâmica a aplicação e sua arquitetura;

**ii.** Dos conceitos estruturais e arquiteturais do sistema operacional Linux, em que foram analisados os trabalhos (artigos, livros e documentações oficiais) que tratam dos principais subsistemas do sistema operacional Linux;

**iii.** Das otimizações do sistema operacional Linux, em que foram analisados os trabalhos (artigos, livros e documentações oficiais) que abordam otimizações específicas do sistema operacional Linux.

Essas três áreas de análise de artigos e bem como outras fontes, oferecem um embasamento geral para alinhar questões de desempenho e sua relevância para este trabalho.

## Contextualização Geral

Em um contexto geral da aplicação Apache Hadoop, quanto às definições, contextualizações e bem como novas perspectivas, (NAIR; MEHTA, 2011), apresenta uma visão geral de clusters da aplicação Apache Hadoop. A abordagem principal do trabalho é quanto à utilização do Apache Hadoop na computação em nuvem e também é apresentada uma visão geral do Apache Hadoop. A computação em nuvem, como serviço, é oferecida sob a tecnologia de virtualização.

O HDFS é abordado no site oficial do Apache hadoop (BORTHAKUR, 2016) e esta referência também é utilizada por outros trabalhos (REZGUI et al., 2014). (RAMOS, 2015), preconiza as justificativas para utilização do Apache Hadoop como promissora solução de código aberto para análise de *Big Data*.

**Otimizações do Sistema Operacional Linux**

(YUANYUAN; PENG, 2010) aborda as métricas para tratar o desempenho em Linux quanto à *CPU-bound* e *I/O-bound*. (DOMINGO; BAILEY, 2016) apresenta uma análise prática de ajustes de parâmetros de configurações no Red Hat Enterprise Linux 6. Esta análise também é útil para distribuição CentOS Linux e alguns parâmetros também são encontrados no Ubuntu Linux LTS. Estas referências foram fundamentais para compreender determinados parâmetros de configurações que não foram embasadas em profundidade pela documentação oficial do kernel do Linux (KERNEL.ORG, 2017).

(BERGNER et al., 2015) apresenta os ajustes de parâmetros de configurações para a linha de servidores IBM, mas amplia a análise para a Máquina Virtual Java (ou JVM - *Java Virtual Machine*), o que é pertinente para o presente trabalho. Utilizando a mesma abordagem, (CILIENDO; KUNIMASA, 2008) e (PARZIALE et al., 2011), apresentam os ajustes de parâmetros de configurações que são relevantes para ambientes virtualizados.

Este trabalho contempla um ambiente de experimentos com virtualização, sendo assim, as referências (BERGNER et al., 2015), (CILIENDO; KUNIMASA, 2008) e (PARZIALE et al., 2011) nos permitiram compreender as particularidades envolvidas quanto aos parâmetros de configurações em ambiente de virtualização.

(ALVES, 2009) apresenta mecanismos de ajustes de parâmetros de configurações nos principais subsistemas do sistema operacional Linux e bem como as técnicas de monitoramento usando utilitários nativos do Linux. Esta é uma obra dedicada a abordar desempenho em Linux. O autor possui vasta experiência em administração de sistemas Linux e o viés prático da obra é de fundamental relevância para este trabalho.

(SILBERSCHATZ; GALVIN; GAGNE, 2015) e (DOMINGO; BAILEY, 2016), abordam conceitos sobre como a troca de contexto pode degradar o sistema em função do algoritmo que realiza o balanceamento de carga das tarefas entre os núcleos, sugerindo uma afinidade de núcleo, mas apresentando também os possíveis efeitos indesejados dessa escolha em arquiteturas NUMA.

(TANENBAUM, 2010) enfoca os conceitos dos sistemas operacionais modernos em multiprocessadores e os principais subsistemas dos sistemas operacionais modernos. O sistema operacional Linux também é abordado como estudo de caso. (TANENBAUM, 2010) desenvolveu o sistema operacional Minix e esta experiência prática no âmbito de sistemas operacionais do autor foi relevante para as construções teóricas abordadas neste trabalho.

Quanto aos aspectos internos do kernel do Linux, Robert Love (LOVE, 2010), aborda os mecanismos internos do Linux para o tratamento de I/O de disco, gerenciamento de memória virtual, arquivos e de tarefas. Esse mesmo autor escreve o capítulo de estudo de caso do sistema operacional Linux em (SILBERSCHATZ; GALVIN; GAGNE, 2015). Especificamente o subsistema de memória virtual do kernel do Linux também é abordado por (GORMAN, 2009).

Christopher Negus (NEGUS; BRESNAHAN, 2015), aborda os conceitos de administração de sistemas Linux, em particular a configuração de desempenho é apresentada de forma generalizada. Seguindo a abordagem de (NEGUS; BRESNAHAN, 2015), em (NEMETH; SNYDER; HEIN, 2010), tratando em uma concepção panorâmica, inclui aspectos de configurações baseados na interface sysctl sob o sistema de arquivos /proc e /proc/sys.

Também, não se tratando diretamente do sistema operacional Linux, mas do sistema operacional FreeBSD, (SOUZA, 2009), lança luz sobre determinados ajustes de parâmetros do kernel, rede e I/O de disco que são pertinentes ao sistema operacional Linux quanto às definições gerais. Nesse sentido, há também uma contribuição notória em (SCHWARTZ; ZAITSEV; TKACHENKO, 2012), especificamente no Capítulo 9, que aborda ajustes de desempenho para o SGBD (Sistema Gerenciador de Banco de Dados) MySQL que consideramos relevantes no que se refere à I/O de disco. Os autores possuem vasta experiência em prover desempenho para o SGBD MySQL utilizando o sistema operacional Linux.

**Otimizações do Apache Hadoop**

No contexto de otimização do Apache Hadoop, (CHO et al., 2013), apresentam melhorias obtidas realizando afinidade de núcleo para tratamento de interrupções de rede e tarefas (processos e threads) com o Apache Hadoop. (LI et al., 2014) realizam modificações diretamente na estrutura do Apache Hadoop para realizar de forma dinâmica alterações em parâmetros para obter melhor desempenho na aplicação. Segundo os autores, há mais de 30 parâmetros no Apache Hadoop que podem ter impacto no desempenho.

(XU; LUO; WOODWARD, 2012) apresentam uma análise de otimização do Apache Hadoop em relação aos níveis de RAID que proporcionam melhor desempenho de *I/O-bound*. Também nesse contexto, (BORTHAKUR, 2016), na documentação oficial do Apache Hadoop, apresenta parâmetros de configurações que permitem obter melhor desempenho no Apache Hadoop.

Consideramos que este referencial teórico foi essencial para que o embasamento teórico das estruturas que compõem um sistema operacional, os subsistemas do kernel do Linux, a visão geral do Apache Hadoop e a visão geral dos parâmetros de configurações, permitissem conduzir à uma pesquisa com solidez em seus aspectos teóricos e nos experimentos práticos.

# 3 TRABALHOS RELACIONADOS

Muitos trabalhos buscam de alguma forma analisar meios que permitem obter melhor desempenho computacional em diversos âmbitos. Isto inclui o sistema operacional Linux. Especificamente dois trabalhos em particular se relacionam diretamente com o cerne deste trabalho. Consideramos estes dois trabalhos pelo fato de que a análise de um dos trabalhos permitiu concluir que ajustes nos parâmetros de configurações (no escalonador de processos do Linux) possuem impacto direto no desempenho.

Por outr lado, outro trabalho, ao analisar o subsistema de I/O de disco do Linux, concluiu que a escolha dos agendadores de I/O de disco e os parâmetros de configurações correlatos não possuem impacto direto no desempenho (melhoria no *throughput*).

Sendo assim, nos alentamos a pesquisar se os ajustes nos parâmetros de configurações do sistema operacional Linux de fato podem melhorar o desempenho do sistema e quais parâmetros de configurações que realmente impactam no desempenho. Nesse sentido, a partir dos resultados apresentados pelos trabalhos analisados, percebemos que deveríamos ampliar o campo de estudo, ou seja, analisar outros subsistemas para entender suas relações particulares.

Além dos parâmetros de configurações nós estudamos e apresentamos neste trabalho outras particularidades que podem influenciar no desempenho. Por exemplo, buscamos compreender a relação do sistema de armazenamento com o agendador de I/O de disco. Também, analisamos alguns problemas causadores de *overhead* e como mitigá-los.

**Análise dos Ajustes de Agendadores de I/O do Linux em *Big Data***

No primeiro trabalho (REZGUI et al., 2014) abordam as relações de desempenho para os quatro principais agendadores de I/O do Linux: ***Anticipatory***, ***Deadline***, **CFQ** e **NOOP**. O trabalho analisa cada um dos agendadores realizando experimentos de carga de trabalho com as ferramentas nativas do Apache Hadoop, mesma abordagem que também segue este trabalho. Atualmente, o agendador de I/O *Anticipatory* não é implementado no kernel do Linux.

**Análise do Impacto de Ajustes das Configurações do Agendador CFS em Sistemas *Multi-Threaded***

(KRISHNAMURTHY; ROUSKAS, 2015) analisam os parâmetros de configurações para obter melhor desempenho em sistemas *multi-threaded* com o protocolo SIP. Os autores analisaram os parâmetros de configurações do escalonador do Linux, o CFS (*Completely Fair Scheduler*), para minimizar a latência nas conexões VoIP (voz sobre IP).

A conclusão em (KRISHNAMURTHY; ROUSKAS, 2015) é que após os ajustes dos parâmetros de configurações do CFS, eles obteveram melhor desempenho no tratamento das conexões VoIP utilizando o protocolo SIP.

(REZGUI et al., 2014) demonstraram que nos experimentos de *workloads* realizados, os quatro agendadores de I/O de disco obtiveram resultados muito próximos e que ajustes nos parâmetros de configurações realizados durante os experimentos não influenciaram nos resultados obtidos. A conclusão inferida é que torna-se necessário o desenvolvimento de novos agendadores de I/O de disco para grandes cargas de trabalho.

Porém, o que é observado por (ALVES, 2009), (DOMINGO; BAILEY, 2016) e (PRATT; HEGER, 2004) é que o agendador de I/O de disco também é dependente das características da aplicação quanto à *CPU-bound* e *I/O-bound* e do sistema de armazenamento subjacente.

Sendo assim, é observado que se o sistema de armazenamento subjacente possui desempenho, então o agendador NOOP é o recomendado entre os agendadores. Portanto, se a operação de Map/Reduce executada é orientada a *I/O-bound* e possui discos de alto desempenho, sugere-se o escalonador NOOP (DOMINGO; BAILEY, 2016).

(ALVES, 2009) realizou experimentos com a criação de um arquivo de aproximadamente 2 GB. O que foi observado é que o *throughput* (em MB/s) com o agendador NOOP foi o melhor caso e considerando que o experimento abordava a escrita de um arquivo, o agendador Deadline obteve o pior desempenho.

Em outro experimento, com amostras baseadas na criação de um arquivo de aproximadamente 2 GB, porém, com processos concorrendo para a escrita do arquivo, o agendador NOOP obteve o pior desempenho, enquanto que o melhor desempenho foi obtido com o agendador CFQ (ALVES, 2009).

(PRATT; HEGER, 2004), observaram diversas variações de desempenho quanto aos agendadores de I/O de disco. Chegaram a conclusão de que existe uma forte correlação entre o agendador de I/O, o perfil da carga de trabalho utilizada, o sistema de arquivos e o desempenho do sistema de armazenamento. Também, ajustes nos parâmetros de configurações foram relevantes no processo da análise.

Outra questão de grande relevância que (PRATT; HEGER, 2004) abordam, é que os resultados variam em função das implementações de RAID. O agendador NOOP obteve melhor desempenho em uma grande estrutura de RAID-0, mas para estruturas menores e RAID-5, o CFQ obteve melhor desempenho. Isso configura nuances em que há relações diretas com a estrutura de armazenamento subjacente, o sistema de arquivos e o tipo de carga de trabalho.

Com as análises, experimentos e resultados que obtivemos, podemos concluir que as contribuições principais deste trabalho são:

- Ampliamos o campo de estudo para outros subsistemas do kernel do Linux. No trabalho de (KRISHNAMURTHY; ROUSKAS, 2015) os autores analisaram o CFS (uma das áreas do gerenciamento de tarefas) com um número reduzido de parâmetros de configurações. No trabalho de (REZGUI et al., 2014) os autores analisaram o subsistema de I/O de disco. Neste trabalho analisamos com maior profundidade o subsistema de tarefas, memória virtual, arquivos e I/O de disco por entendermos que tais subsistemas são fortemente relacionados (STUART, 2010).

- Catalogamos os principais parâmetros de configurações que de fato influenciam no desempenho do sistema operacional Linux. Para cada parâmetro, a partir das análises das cargas de trabalho, obtivemos ajustes que produziram melhor desempenho que as configurações nativas do sistema operacional Linux. Esta catalogação de parâmetros de configurações com os melhores resultados podem favorecer empresas (e outros segmentos) a proverem melhor desempenho para aplicações de misssão crítica.

- Em nossas análises também nos dedicamos a compreender a semântica dos parâmetros de configurações. Isto permite lançar luz para uma correta compreensão do parâmetro de configuração e como este pode impactar no desempenho caso seja configurado de forma inadequada.

A análise dos parâmetros de configurações do sistema operacional Linux favorece também que se conheça o impacto em outras aplicações de missão crítica em uma infraestrutura de tecnologia da informação com características semelhantes de *CPU-bound* e *I/O-bound* do Apache Hadoop.

# 4 VISÃO GERAL DO SISTEMA OPERACIONAL LINUX E DO APACHE HADOOP

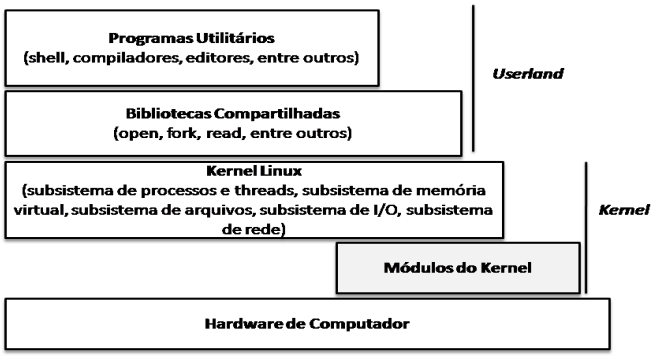
Este capítulo tem como princípio descrever de forma particular os dois objetos macros de estudo que permeiam o presente trabalho. São apresentados os pontos e as conceituações gerais quanto ao sistema operacional Linux e bem como a aplicação principal utilizada neste trabalho; o Apache Hadoop.

## 4.1 O Sistema Operacional Linux

O sistema operacional Linux foi idealizado e desenvolvido por Linus Torvalds, diretamente inspirado a partir do MINIX, sistema operacional este, desenvolvido por Andrew S. Tanenbaum (TANENBAUM, 2010).

A figura 1, apresenta uma visão estrutural do sistema operacional Linux. A figura foi idealizada a partir de (SILBERSCHATZ; GALVIN; GAGNE, 2015) e (TANENBAUM, 2010).

Figura 1 - Estrutura do sistema operacional Linux.




A partir da figura 1, este trabalho concentrou exclusivamente na área do kernel do Linux, em que cada subsistema oferece uma interface (parâmetros de configurações) para controlar o comportamento do kernel. O corpo de código do kernel do Linux implementa o gerenciamento de processos e *threads* (ou tarefas), memória virtual, sistema de arquivos e I/O de disco e rede, dispositivos, entre outros.

### 4.1.1 *Subsistema de Tarefas (Processos e Threads)*

O Linux é um sistema operacional multitarefa e possui suporte a hardware com múltiplos processadores. Quanto às *threads*, o Linux implementa o modelo um-para-um, ou seja, para uma *thread* da *userland* (espaço do usuário) é mapeada uma *thread* do kernel. Processos e *threads* são denominados **tarefas** (SILBERSCHATZ; GALVIN; GAGNE, 2015).

No que se refere ao gerenciamento de tarefas no Linux, para este trabalho, três abordagens são primariamente relevantes para o objeto de estudo:

(i). O gerenciamento de tarefas quanto à arquitetura, no caso NUMA (*Non Uniform Memory Access* – Acesso não-uniforme à memória);

(ii). O gerenciamento de tarefas quanto à arquitetura, no caso UMA (*Uniform Memory Access* – Acesso uniforme à memória) e;

(iii). O gerenciamento de tarefas quanto ao modelo, no caso SMP (*Symmetric Multiprocessing* – Multiprocessamento simétrico).

Tanto as arquiteturas quanto o modelo são amplamente utilizados nos computadores atuais e suportados pelo kernel do Linux. A arquitetura NUMA é oferecida, por exemplo, por meio da tecnologia Intel QPI (*QuickPath Interconnect*) ou *HyperTransport* em processadores AMD (DOMINGO; BAILEY, 2016). Também é muito comum, quanto à **arquitetura** (STALLINGS, 2010), o acesso simétrico à memória, em que o acesso à memória em qualquer região da memória física é aproximadamente o mesmo para cada processador (STALLINGS, 2010).

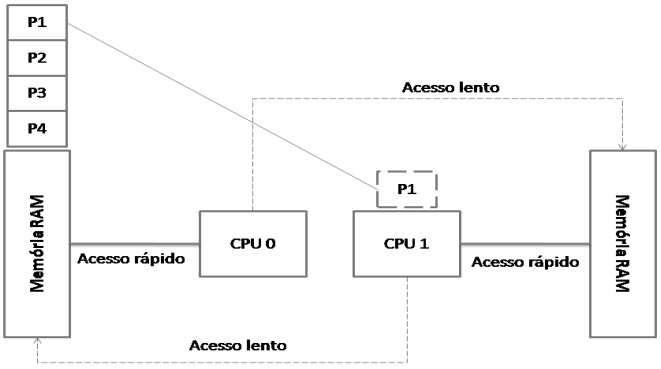
Há que se observar que aqui fazemos distinção quanto à arquitetura, tratando-se de memória compartilhada SMP (*Shared-Memory Processing*) e quanto ao modelo, trantando-se de multiprocessamento simétrico SMP (*Symmetric Multiprocessing*). O **modelo** SMP trata da capacidade de um sistema operacional ser executado em qualquer processador disponível ou em vários processadores disponíveis (STALLINGS, 2010).

Atualmente, o Linux possui melhorias fundamentais no tratamento de multiprocessamento simétrico, principalmente quanto à afinidade de processador, algoritmos de balanceamento de carga e melhorias no suporte a tarefas interativas (SILBERSCHATZ; GALVIN; GAGNE, 2015).

Um aspecto importante no âmbito de sistemas operacionais é o escalonamento de processos, que consiste de um mecanismo do kernel para escolher a próxima tarefa que irá receber a CPU (STUART, 2010). A escolha da tarefa requer vincular a CPU entre estas e tal mecanismo é conhecido como **troca de contexto** (STUART, 2010). É necessário salvar a configuração do contexto da tarefa atual e posteriormente restaurar tal configuração (deve haver suporte da CPU). A troca de contexto é concebida como um *overhead* (carga extra de trabalho), pois nenhum trabalho útil é realizado (SILBERSCHATZ; GALVIN; GAGNE, 2015).

A troca de contexto acentua o *overhead* quando processos com seus respectivos estados salvos que estavam executando em uma determinada CPU são restaurados para serem executados em uma CPU distinta. Em aplicações que requerem *CPU-bound* ou *I/O-bound*, isso pode impactar no desempenho. O Linux utiliza as chamadas de sistema **prefetch()** e **prefetch_stack()** para melhorar o desempenho de cache durante a troca de contexto. Essas instruções requerem suporte da CPU (STUART, 2010). Em arquiteturas NUMA pode-se ainda agravar o *overhead*, como apresentado na figura 2, adaptada de (SILBERSCHATZ; GALVIN; GAGNE, 2015):

Figura 2 - Acesso remoto a memória em arquitetura NUMA.



Em uma arquitetura NUMA, um determinado processo que reside na memória de acesso rápido por uma CPU pode ser executado em uma CPU (ou núcleo) estrangeira, em que o acesso é lento (SILBERSCHATZ; GALVIN; GAGNE, 2015). A **afinidade de processador** é uma técnica que permite ao administrador de sistemas ou programador, planejar estrategicamente, que uma determinada tarefa seja executada somente na CPU 0, a interface de rede tenha suas requisições atendidas somente na CPU 1, entre outros arranjos. Esta abordagem mitiga o *overhead* da troca de contexto entre processadores distintos quanto à alocação de CPU estrangeira em arquiteturas NUMA.

Há algoritmos para realizar o balanceamento de carga dos processos entre as CPUs, de forma que determinadas CPUs não fiquem subutilizadas enquanto outras estejam superutilizadas. No Linux, o algoritmo de balanceamento de carga é executado a cada 200 milissegundos, porém, o balanceamento de carga termina por antagonizar os benefícios da afinidade de processador (SILBERSCHATZ; GALVIN; GAGNE, 2015).

Neste trabalho consideramos relevante este tópico sobre arquitetura NUMA pelo fato de aplicações críticas também serem executadas em ambientes que não contemplam virtualização. A tecnologia de QPI e *HyperTransport* são amplas nos ambientes de computação e, em nossas análises, verificamos que a ferramenta **numad** (nativa em muitas distribuições Linux) é a solução recomendada para evitar execução de tarefas em memória estrangeira (ou remota).

### 4.1.2 *Subsistema de Memória Virtual*

A unidade de gerenciamento de memória (MMU - *Memory Management Unit*) trata as palavras da memória principal como páginas físicas. O kernel do Linux trata tais páginas como unidades fundamentais para o gerenciamento de memória (LOVE, 2010). O tamanho da página é definido a partir da arquitetura de hardware, mas majoritariamente o tamanho de página de uma arquitetura x86 é de 4 KB, enquanto que arquiteturas x86_64 é de 8 KB. Tais páginas são representadas no kernel, como descrito no arquivo "linux/mm.h", pela estrutura *struct page*.

O Linux realiza a modelagem da memória principal (física) dividindo-a em três zonas principais nas arquiteturas de CPU Intel x86: ZONE_DMA, ZONE_NORMAL e ZONE_HIGHMEM (SILBERSCHATZ; GALVIN; GAGNE, 2015). Estas zonas são representadas no arquivo "linux/mmzone.h"pela estrutura *struct zone*.

O espaço de endereços é divido em duas ordens: **espaço de endereços do kernel** e **espaço de endereços do usuário**. O espaço de endereços do usuário inicia no endereço 0 e o tamanho máximo é dependente da arquitetura, definido em TASK_SIZE, no arquivo "include/asm/processor.h". Acima do limite definido em TASK_SIZE, encontra-se o espaço de endereços do kernel (BOVET, 2005).

Em arquitetura x86_64 ou IA64, o espaço de endereços é representado por oito regiões de tamanhos iguais, sendo as regiões entre 0 e 4 (ambos inclusivos) definindo o espaço de endereços do usuário e as regiões entre 5 e 7 (ambos inclusivos) definindo o espaço de endereços do kernel. Em arquitetura NUMA o kernel segmenta a memória física em diversos nós.

### 4.1.3 *Subsistema de I/O de Disco e Sistema de Arquivos*

O subsistema de I/O no Linux é implementado majoritariamente por meio de *drivers* de dispositivo (TANENBAUM, 2010). Nesse contexto são utilizadas as abordagens das metades inferiores (*bottom halves*) e metades superiores (*top halves*). A metade inferior trata das sub-rotinas de interrupção e é utilizada em tempo crítico e faz a interface com o restante da estrutura do Linux. A metade superior executa no contexto do kernel e interage com o dispositivo (LOVE, 2010).

Especificamente quanto à I/O de disco, área de relevância para este trabalho, o Linux oferece suporte à solicitação de bloco, a qual constitui-se de duas áreas principais: (i) as funções que gerenciam filas de requisições e (ii) os agendadores de I/O. O Linux possui três agendadores de I/O: **NOOP**, **Deadline** e o **CFQ** (STUART, 2010).

A função do agendador de I/O de disco é reorganizar ou agrupar as solicitações de leitura/escrita nos dispositivos de bloco (TANENBAUM, 2010). Como observado em (ALVES, 2009) e (PRATT; HEGER, 2004), a escolha do agendador de I/O de disco implica em impactos consideráveis no desempenho das aplicações.

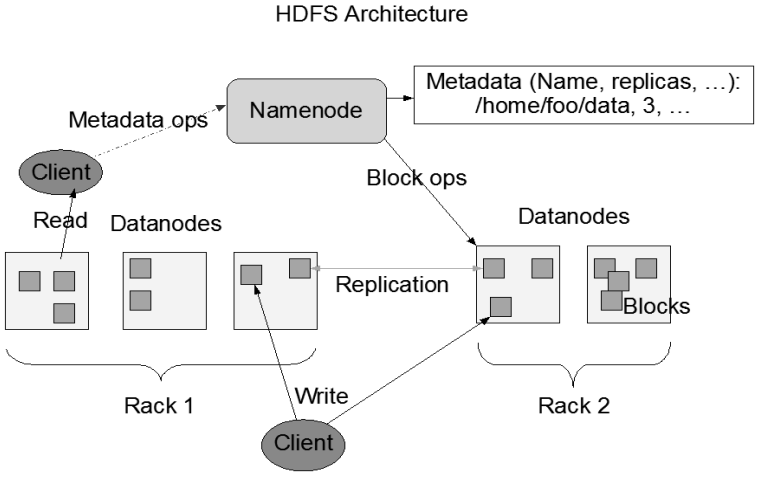
O Linux oferece suporte a diversos sistemas de arquivos por meio da interface VFS (*Virtual File System*). Os sistemas de arquivos incluem RaiserFS, Ext2, Ext3, Ext4, entre outros. Cada partição pode ser formatada com um sistema de arquivos distinto. A escolha do sistema de arquivos é crucial para obter desempenho em função dos algoritmos de análise de percurso de caminho (*lookup path*) e tradução de bloco lógico para bloco físico no disco (STUART, 2010).

Cada sistema de arquivos possui características que permitem alterar como as operações de leitura/escrita ocorrem (síncrono e assíncrono) e tais características são representadas por meio de diretivas do sistema de arquivos. Nesse aspecto, há um conflito, pois por um lado obtém-se melhor desempenho nas operações de leitura/escrita e por outro, a confiabilidade é diminuída, o que para aplicações que manipulam dados críticos, a perda de dados pode impactar nos negócios.

## 4.2 O Apache Hadoop

O Apache Hadoop faz parte de um projeto de código aberto para computação distribuída confiável e escalável. O Apache Hadoop utiliza as funções *Map* e *Reduce* para suas operações em cluster. A figura 3 apresenta a arquitetura geral do Apache Hadoop:

Figura 3 - Visão geral da arquitetura do Apache Hadoop




**Fonte:**http://hadoop.apache.org

Na figura 3, o *Namenode* e o *Datanode* são as partes de software executadas em máquinas ditas *commodities*, ou seja, máquinas de custo relativamente baixo ou virtualizadas. O HDFS (*Hadoop Distributed File System*), um dos objetos fundamentais do Apache Hadoop, foi desenvolvido usando a linguagem de programação Java, o que permite a portabilidade, ou seja, qualquer hardware com a JVM pode executar o *Namenode* ou *Datanode*.

Uma arquitetura em cluster envolve a implementação do HDFS de forma que uma máquina principal execute o *Namenode* e os outros nós do *cluster* processam uma instância do *Datanode*, esta abordagem simplifica a arquitetura. O NameNode é o responsável por arbitrar e também é o repositório para todos os metadados do HDFS.

O HDFS é um sistema de arquivos distribuídos, como o GFS, AFS e NFS, porém, o cerne do projeto do HDFS é a tolerância a falhas e requisitos de engenharia para operação em hardware de baixo custo. Mesmo com aderência ao POSIX, o HDFS estende alguns requisitos do POSIX para permitir o acesso em fluxo contínuo aos dados do sistema de arquivos.

O Apache Hadoop é uma aplicação orientada a *CPU-bound*, pelo fato de tratar grande computação e realizar uso intenso da memória por meio dos algoritmos para tratamento massivo de dados e também é uma aplicação orientada a *I/O-bound* pelo fato da alta carga de leitura e escrita de dados de forma distribuída ou local.

# 5 METODOLOGIA

Adotamos uma metodologia que assemelha-se ao modelo utilizado por (KRISHNAMURTHY; ROUSKAS, 2015). Neste sentido, comparamos o desempenho obtido com os parâmetros de configurações que foram modificados (neste trabalho, denominado **Modelo Ajustado**) com o desempenho dos parâmetros de configurações nativos do sistema operacional Linux (neste trabalho, denominado **Modelo Comum**).

Com o objetivo de obtermos um ambiente de experimentos contemplando *multi-threaded*, uso itensivo de CPU e de I/O de disco, optamos pelo Apache Hadoop como aplicação principal. O Apache Hadoop é utilizado em um dos trabalhos relacionados (REZGUI et al., 2014) e sua caracterítica de *multi-threaded* também (KRISHNAMURTHY; ROUSKAS, 2015).

Para realizarmos as cargas de trabalho nos experimentos envolvendo I/O de disco, utilizamos o TestDFSIO. O TestDFSIO é uma escolha coerente para determinar a eficiência dos agendadores de I/O de disco do Linux (REZGUI et al., 2014). Também utilizamos o TIOBench, software listado no portal OpenBenchmarking.Org para realizar cargas de trabalho de I/O de disco com múltiplas *threads*. Este software possui granularidade para execução das cargas de trabalho no quesito de número de *threads* concorrentes, tamanho de bloco, tamanhos de arquivos e quantidade de arquivos.

No sentido de análise *multi-threaded* do comportamento do sistema durante a execução do Apache Hadoop, utilizamos o código do WordCount do site oficial do Apache Hadoop (https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html). Com a execução do código do WordCount, utilizamos os utilitários *perf* e *time* (/usr/bin/time). Estes utilitários foram utilizados para análises envolvendo a troca de contexto e latências particulares do agendador de processos do Linux. A tabela 1 apresenta um resumo dos softwares utilizados.

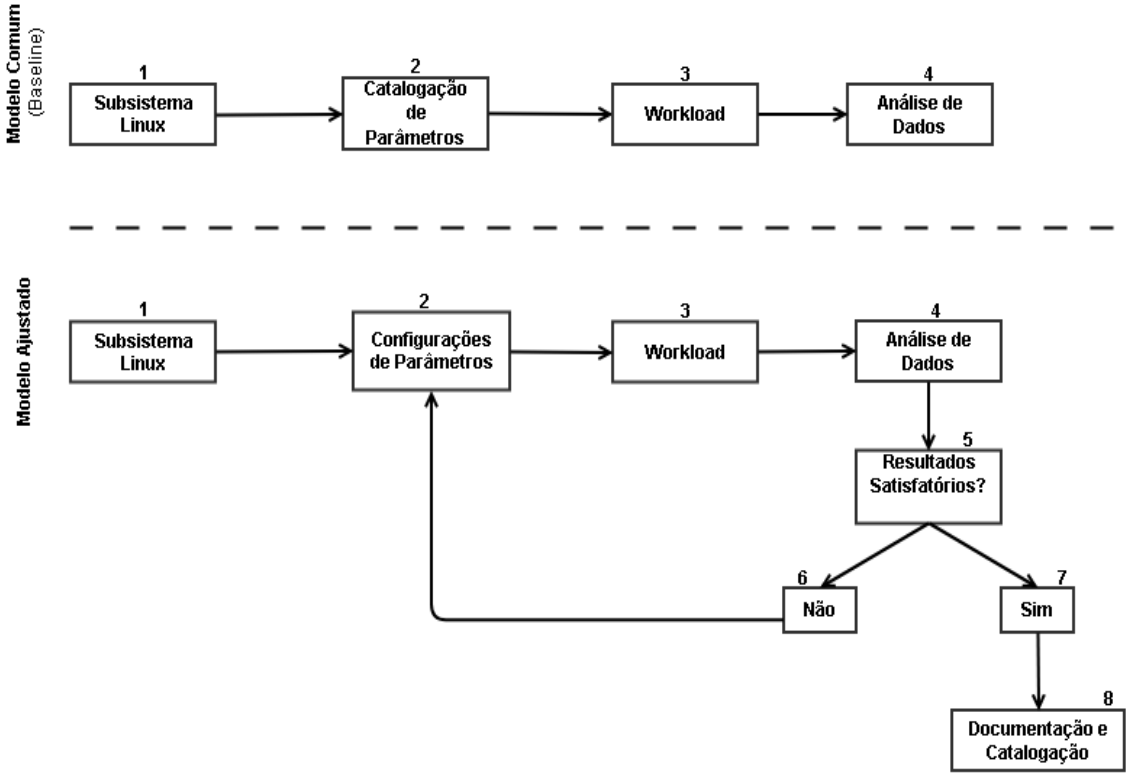
Tabela 1 - Softwares utilizados como parte da metodologia

| **Software** | **Versão** | **Descrição** |
|---|---|---|
| TestDFSIO | 2.7.3 | Realização de cargas de I/O de disco. |
| TIOBench | 1.2.0 | Realização de cargas de I/O de disco. |
| WordCount | - | Código para análise de ambiente *multi-threaded*. |
| time | GNU time 1.7 | Obtenção de dados de execução da carga de trabalho em ambiente *multi-threaded*. |
| perf | 4.4.40 | Obtenção de dados de execução da carga de trabalho em ambiente *multi-threaded*. |
| strace | 4.11-1 | Obtenção de dados de execução da carga de trabalho em ambiente *multi-threaded*. |
| cgroup-tools | 0.41-7 | Obtenção de dados de execução da carga de trabalho em ambiente *multi-threaded*. |

A utilização da carga de trabalho com o TestDFSIO e WordCount como parte da metodologia em análises do comportamento do sistema operacional Linux também podem ser encontradas em outros trabalhos (FUJISHIMA; YAMAGUCHI, 2016), (LI et al., 2014), (WANG; LIN; TANG, 2012).

A figura 4, apresenta um diagrama esquemático macro da metodologia utilizada para este trabalho:

Figura 4 - Diagrama esquemático macro da metodologia proposta.




Este diagrama foi idealizado a partir de (ALMEIDA; MENASCE, 2003), que apresenta uma metodologia para realização de testes de desempenho. Neste sentido, consideramos relevantes na metodologia de (ALMEIDA; MENASCE, 2003) para nossos propósitos a **"definição dos objetivos do teste"**, que neste trabalho foi realizado por meio da seleção de um dos subsistemas do kernel do Linux (tarefas, memória virtual, arquivos e I/O de disco) para determinar os parâmetros de configurações que impactam no desempenho.

Utilizamos as etapas de **"definir a carga de trabalho"**, **"configurando o ambiente de teste"** e **"executando os testes"** para seleção e execução do TestDFSIO, TIOBench e o código do WordCount aplicados para os subsistemas do kernel do Linux.

Realizamos as análises dos resultados tanto no Modelo Comum (parâmetros de configurações nativos do sistema operacional Linux) quanto no Modelo Ajustado (parâmetros de configurações ajustados). Em nossa metodologia consideramos a relevância de uma abordagem cíclica para obtermos outros resultados modificando os parâmetros de configurações.

Sendo assim, começaremos a detalhar o diagrama da figura 4. No **Modelo Comum** (parâmetros de configurações nativos do sistema operacional Linux), é adotado um subsistema do kernel do Linux (tarefas, memória virtual, arquivos e I/O de disco) para análise **(1)**. Esta análise consiste em realizar um levantamento dos principais parâmetros de configurações de cada subsistema, privilegiando os parâmetros de configurações que produzem impacto no desempenho **(2)**. O referencial teórico e os trabalhos relacionados foram cruciais nesta etapa.

Então são realizadas as cargas de trabalho relativas ao subsistema em análise por meio das ferramentas descritas na tabela 1 **(3)** e assim, os dados são analisados e catalogados **(4)**.

No **Modelo Ajustado** (parâmetros de configurações ajustados), é adotado um subsistema do kernel do Linux (tarefas, memória virtual, arquivos e I/O de disco) **(1)**, realizamos ajustes nos parâmetros de configurações individualmente, ou seja, analisamos e testamos cada parâmetro de configuração **(2)**, então, a carga de trabalho foi aplicada **(3)** e os dados foram analisados **(4)**. Quando os resultados foram satisfatórios, ou seja, obtivemos melhor desempenho a partir dos valores testados com a carga de trabalho **(5)(7)**, os parâmetros de configurações que proporcionaram melhor desempenho foram documentados e catalogados **(8)**. Caso contrário, reajustamos os parâmetros até que resultados melhores que o do modelo Modelo Comum fossem alcançados **(6)(2)**.

Os detalhes envolvendo as razões práticas dos ajustes estão descritas nos capítulos posteriores particulares de cada subsistema (Capítulo 6, Capítulo 7 e Capítulo 8). Mais informações também estão descritas no capítulo de análise dos resultados (Capítulo 9). No apêndice estão catalogados todos os parâmetros analisados neste trabalho.

Quando um subsistema do Linux é definido e aplicado o ajuste no parâmetro de configuração, questões como a quantidade de memória, a quantidade de núcleos, as rotações por minuto do disco, o tipo do disco (SSD, HDD) e o protocolo do disco (PATA, SATA, SAS), foram critérios cruciais a serem observados para a fase de experimentos.

Neste sentido, observamos que poucos núcleos de CPU podem impactar no tempo de execução, que pouca memória pode envolver constante utilização de memória de troca (*swap*), que discos com baixa rotação por minuto (RPM) podem impactar no acesso ao disco, que discos SSD possuem características que impactam no agendador de I/O de disco e que o protocolo do disco e a tecnologia do disco podem aumentar a latência.

Na fase de experimentos foi aplicada a carga de trabalho. Esta carga de trabalho consiste da execução e análise com as ferramentas dispostas na tabela 1. Após a carga de trabalho, os resultados obtidos foram analisados e caso o resultado fosse ótimo, ou seja, obtivemos melhor desempenho com o ajuste proposto, então geramos a catalogação com os resultados obtidos e estão descritos na análise dos resultados (capítulo 9).

Cada parâmetro de configuração foi analisado individualmente, porém, seletivamente e sucessivamente. O que queremos dizer com isso é que a partir do momento que conseguimos obter o melhor resultado de um determinado parâmetro, este foi mantido e analisado o próximo parâmetro de configuração, buscando um modelo de desempenho incremental.

As ferramentas do Apache Hadoop fornecem dados quanto à latência, tempo decorrido de leitura/escrita em disco, tempo de execução e outros dados. Esses dados também foram utilizados como parâmetros para comparação do modelo de pré-ajuste (Modelo Comum) e pós-ajuste (Modelo Ajustado).

Os dados coletados (parâmetros de configurações) antes da implementação dos ajustes de desempenho foi denominado **Modelo Comum - Subsistema X**, sendo X, um subsistema do sistema operacional Linux. Os dados foram obtidos a partir das execuções das cargas de trabalho, sendo o mínimo de cinco execuções (uma única execução não consideraria estruturas de cache construídas por execuções sucessivas, o que poderia implicar em dados inconsistentes).

Após a aplicação da carga de trabalho os dados eram coletados para análise. O sistema operacional foi reiniciado a cada aplicação da carga de trabalho.

Os dados coletados (parâmetros de configurações) após o ajuste de desempenho foi denominado **Modelo Ajustado - Subsistema X**, sendo X, um subsistema do sistema operacional Linux. Os dados foram obtidos a partir das execuções das cargas de trabalho, sendo o mínimo de cinco execuções (uma única execução não consideraria estruturas de cache construídas por execuções sucessivas, o que poderia implicar em dados inconsistentes).

Após a aplicação da carga de trabalho os dados eram coletados para análise. O sistema operacional foi reiniciado a cada aplicação da carga de trabalho.

Majoritariamente, os parâmetros de configurações são ajustados por meio do utilitário **sysctl** (*system control*) que atua diretamente no sistema de arquivos /proc ou na subestrutura /proc/sys para configurações de parâmetros do kernel durante a execução ou para serem aplicadas após a inicialização do sistema.

Na tabela 2 (de exemplo), apresentamos os dados catalogados para as inferições do subsistema de memória virtual concernente ao parâmetro de configuração `vm.swapness` para o Modelo Comum, ou seja, com valor de configuração padrão do sistema.

Tabela 2 - Modelo Comum - Subsistema de Memória Virtual.

| **Parâmetro** | **Valor** | **Descrição** | **Tempo Execução - WordCount (em segundos)** |
|---|---|---|---|
| `vm.swaapness` | 60 | Determina propensão à relização de swap | 17,8 |

Outras colunas com informações pertinentes estarão disponíveis dependendo do subsistema. As tabelas de exemplo aqui dispostas estão condensadas.

Na tabela 3 (de exemplo), apresentamos dados catalogados para a inferição do subsistema de memória virtual concernente ao parâmetro de configuração `vm.swapness` para o Modelo Ajustado, ou seja, com valor de configuração concernente ao ajuste de desempenho proposto.

Tabela 3 - Modelo Ajustado - Subsistema de Memória Virtual.

| **Parâmetro** | **Valor** | **Descrição** | **Tempo Execução - WordCount (em segundos)** |
|---|---|---|---|
| `vm.swaapness` | 10 | Determina propensão à relização de swap | 14,1 |

Neste exemplo, observamos que a redução da propensão a *swap*, permitiu uma melhoria no tempo de execução de 3,7 segundos para um experimento realizado com o WordCount.

Um comando nativo do sistema, o sysctl, com filtros aplicados, podemos observar que há aproximadamente novecentas (900) variáveis ajustáveis, sendo aproximadamente quarenta (40) parâmetros relacionados ao subsistema de memória virtual, duzentos e quatorze (214) parâmetros relacionados a características do kernel e quinhentos e setenta e três (573) parâmetros pertinentes ao subsistema de rede (não analisado neste trabalho). O resultado da saída com o comando sysctl pode variar de acordo com a versão do kernel e distribuição Linux.

Também, nossa proposta foi, a partir desse conjunto de parâmetros ajustáveis, encontrar um subconjunto máximo de parâmetros que impactam no desempenho do sistema operacional Linux. Outras diretivas sob a estrutura /proc e /sys também foram analisadas e serão abordadas neste trabalho.

## 5.1 **O Ambiente de Experimentos**

Em nosso ambiente de experimentos utilizamos a virtualização com o software VMWare Advanced Edition 4.1. Para esse ambiente foi disponibilizado um Processador Intel® Xeon® CPU E5630 2,53GHz (com 4 núcleos), com 8 GB de Memória RAM e 40 GB de armazenamento em disco exclusivo para a partição do Apache Hadoop (/hadoop). A máquina com o *Hypervisor* consistia de um computador Dell R710 com dois processadores Intel® Xeon® CPU E5630 2,53GHz com quatro núcleos cada processador.

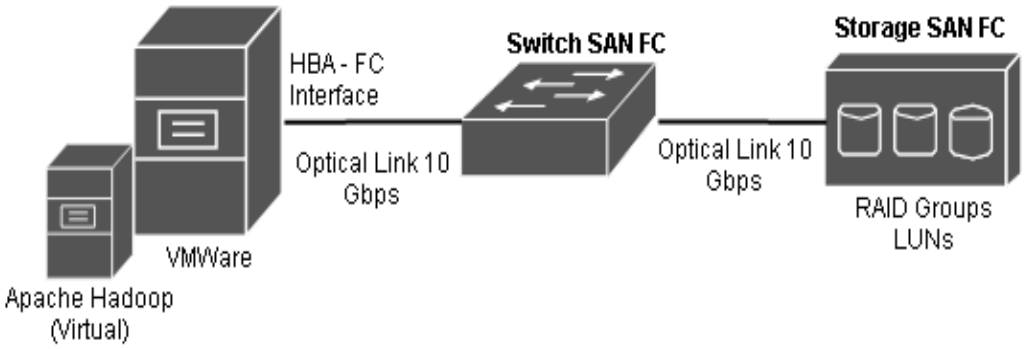
No ambiente de experimentos foi utilizado o sistema operacional Linux Ubuntu Server 14.4 de 64 Bits e versão do kernel 3.13. Foi utilizada a versão 2.7.3 do Apache Hadoop e versão do Java 1.7.0_79.

O subsistema de armazenamento do ambiente de virtualização utiliza uma rede SAN com discos SAS 15K RPM. No ambiente de virtualização o *hypervisor* VMWare é conectado ao switch SAN FC (*Fibre Channel*) por meio de um enlace óptico de 10 Gbps. O Storage possui RAID Groups com níveis de RAID em RAID-10 e RAID-5. Os RAID Groups são compostos por discos SAS 15K RPM, Small Factor de 300 GB.

Foi criada uma LUN específica e DataStore específico para máquina virtual do Apache Hadoop. Todas as operações de I/O de disco utilizaram esta infraestrutura SAN FC. Não foram utilizados discos locais no ambiente de experimentos.

Na figura 5 apresentamos um diagrama contendo as conexões de rede e infraestrutura pertinentes ao ambiente de experimentos utilizando virtualização.

Figura 5 - Infraestrutura do ambiente de experimentos contemplando virtualização.




O primeiro objeto à esquerda representa a máquina virtual Linux com o Apache Hadoop. Esta infraestrutura utiliza o protocolo FC (*Fibre Channel*) para o ambiente da rede SAN. O Servidor *hypervisor* VMWare possui uma interface de 1 GE para comunicação com a rede LAN Ethernet.

A escolha da virtualização como frente de experimentos foi considerada uma vez que tal tecnologia é consolidada no âmbito corporativo e na computação em nuvem.

Não foram realizados ajustes de parâmetros de configurações quanto à melhoria de desempenho no Apache Hadoop, no HDFS ou na JVM (*Java Virtual Machine*). Também, neste ambiente de experimentos o Apache Hadoop foi configurado como ***standalone*** (ou *single node*).

Também, assim como nos trabalhos relacionados e parte da litetura contida no referencial teórico, optamos por uma **pesquisa qualitativa**, a qual não requer o uso de métodos e técnicas estatísticas (PRODANOV; FREITAS, 2013), mas os dados são analisados indutivamente com teor mais descritivo e requer trabalho mais intensivo (PRODANOV; FREITAS, 2013).

Devido ao número de parâmetros de configurações a serem experimentados bem como a variabilidade de valores possíveis a serem ajustados, nos embasamos primeiramente em conhecer, por meio do referencial teórico, trabalhos relacionados e outras fontes, quais parâmetros que impactam no desempenho do sistema. Esta abordagem permitiu que os parâmetros de configurações fossem catalogados.

Cada parâmetro de configuração possui valores que podem ser ajustados para otimizar o sistema operacional Linux. Por exemplo, ao parâmetro nr_requests podemos atribuir valores variando entre 1 a 32768. Isto conduziria a um conjunto de experimentos, considerando cada parâmetro de configuração a ser analisado e a gama de valores possíveis ajustados, inviáveis.

Sendo assim, tivemos dois fatores fundamentalmente decisivos: **(i)** a escolha assertiva dos parâmetros de configurações a serem analisados e **(ii)** os valores a serem ajustados. Neste sentido, a correta e coerente escolha das fontes a serem analisadas permitem conhecer assertivamente os parâmetros de configurações que impactam no desempenho e também a direção quanto aos valores a serem ajustados em tais parâmetros de configurações.

Portanto, ao realizarmos experimentos quanto aos valores a serem ajustados nos parâmetros de configurações, em nossa metodologia, nos orientamos primeiramente por meio das referências adotadas. Então, fizemos outras comparações considerando o valor padrão do parâmetro de configuração. Por exemplo, o valor padrão do parâmetro nr_requests é 128. Um dos valores recomendados pelas referências é 256. Com esses valores como base, realizamos experimentos com o valor 256, porém, adotamos como métrica analisar também alguns de seus múltiplos para comparações com os resultados dos valores recomendados nas referências e no Modelo Comum.

Nesse sentido, por exemplo, fizemos experimentos quanto ao parâmetro de configuração nr_requests com os valores ajustados em 64, 128, 256 e 512. Um dos objetivos é construir o Modelo Ajustado a partir de um conjunto de parâmetros de configurações e valores ajustados que obtiveram melhor desempenho. Portanto, valores de parâmetros de configurações que quando ajustados conduziram a um desempenho inferior ao Modelo Comum, ou seja, com o valor padrão do sistema, tais valores não foram catalogados e os experimentos foram constituídos em no máximo de três execuções.

# 6 O SUBSISTEMA DE TAREFAS E A CPU

Os processos e *threads* (ou tarefas, usados aqui como sinônimos) são o cerne do sistema operacional. As principais chamadas de sistema para criação de processos no Linux são fork(), vfork() e clone(), sendo esta última não aderente ao POSIX (TANENBAUM, 2010).

O Linux possui uma estrutura para manter as informações dos processos do sistema chamada struct task_struct *task, que é relativamente grande, possui aproximadamente 1.7 KB, mas contém todas as informações necessárias do processo para o kernel (LOVE, 2010). Entre as informações, incluem a prioridade do processo, fatia de tempo (*slice time*), ID do processo, registradores do processo, arquivos abertos, espaço de endereços do processo, estado do processo e processos filhos (STUART, 2010).

Este capítulo aborda exclusivamente o subsistema de tarefas e os seus principais pontos de análises relevantes.

## 6.1 Descritores de Arquivo

Antes de um determinado arquivo ser lido/escrito o arquivo deve ser aberto. As permissões e outras características do arquivo são verificadas e caso sejam satisfeitas, o arquivo será aberto e suas operações realizadas. O arquivo quando é aberto, o sistema operacional grava um inteiro para representar este arquivo e este é chamado de **descritor de arquivo** (TANENBAUM, 2010). Há uma tabela de descritores de arquivos gerenciada pelo kernel.

Uma aplicação com grande carga de trabalho manipula entre centenas e milhares de arquivos dependendo da análise de dados à qual se destina. O número de arquivos abertos no sistema tem limites e caso este limite seja alcançado, mensagens de erro são emitidas no sistema e a aplicação pode gerar falhas.

Geralmente, o padrão do sistema quanto ao número de arquivos abertos são de 1024. Alguns trabalhos recomendam definir valores entre 4096 e 65536. Nos experimentos esse parâmetro foi ajustado para todos os usuários do sistema, porém, com valor maior que o recomendado pelas melhores práticas (665535) (DOMINGO; BAILEY, 2016).

## 6.2 **Afinidade de Processador**

Na afinidade de processador é uma das técnicas para mitigar o *overhead* da troca de contexto (SILBERSCHATZ; GALVIN; GAGNE, 2015). Sendo assim, duas abordagens são importantes de se considerar: **SMP** e **NUMA**. O acesso remoto a memória (*hop*) impõe uma determinada penalidade, definida como 2 (N + $M_t$), sendo N a penalidade por salto e $M_t$ o tempo de acesso a memória (DOMINGO; BAILEY, 2016).

Dada tal penalidade, as aplicações que requerem desempenho são afetadas pelo acesso regular à memória remota na topologia NUMA durante a troca de contexto (DOMINGO; BAILEY, 2016).

Para realizar a afinidade de processador e minimizar o *overhead* da troca de contexto há três abordagens possíveis (DOMINGO; BAILEY, 2016):

(i). Ajuste estático, em que os valores são configurados no sistema de arquivos ProcFS. Isto requer atribuir manualmente valores em determinados objetos do sistema de arquivos ProcFS para indicar ao kernel que determinado PID (ou IRQ) deve ser executados em processadores (ou núcleos) específicos;

(ii). Por meio do utilitário taskset, para configuração estática de linha de comando. Este utilitário permite a realização de afinidade de processador por linha de comando, sem intervenção manual em objetos do sistema de arquivos ProcFS;

(iii). Por meio de configuração dinâmica e automatizada, com o *daemon* numad.

Neste trabalho, para fins de automatização dos experimos, utilizamos a abordagem de ajuste estático quanto à SMP baseado na arquitetura UMA, com **acesso uniforme à memória**. Neste sentido, foi desenvolvido um código que faz uso das funções CPU_ZERO (realiza um *flush* da estrutura em memória) e CPU_SET (realiza a afinidade de processador) para realizar a afinidade de processador. Uma função verificava os PIDs dos processos JAVA e realizava a afinidade de processador.

A afinidade de IRQ da interface de rede foi realizada com a função set_irq_affinity(), dedicando parte dos núcleos para IRQ. A ferramenta taskset não garante que o acesso remoto em arquitetura NUMA não ocorra (DOMINGO; BAILEY, 2016). Sendo assim, para arquitetura NUMA, observamos que a melhor opção é a utilização do daemon numad, principalmente em ambientes que não contemplam virtualização.

Uma vez que o Apache Hadoop é uma aplicação baseada em *threads*, foi construída uma estratégia de afinidade de processador para garantir um determinado número de nós específicos (nó UMA) para as *threads*, o que evita a substituição de linha de cache, problema conhecido como *cache trashing* (DOMINGO; BAILEY, 2016).

Por outro lado, se a afinidade for realizada para um único núcleo e as *threads* acessarem os mesmos dados armazenados em cache, o desempenho pode ser melhor (DOMINGO; BAILEY, 2016), porém, não é provável tal ocorrência se determinada aplicação crítica executa as *threads* em múltiplos núcleos.

## 6.3 **Escalonadores de CPU**

O escalonador de tarefas do sistema operacional Linux constitui-se de duas grandes políticas (DOMINGO; BAILEY, 2016):

(i). **Política de Escalonadores de Tempo Real**. Utilizados para aplicações de tempo real e esta, possui precedência sobre outras aplicações. Os principais modelos para tempo real são **SCHED_FIFO** e **SCHED_RR**.

(ii). **Política de Escalonadores de Tempo Normal**. Utilizado para todas as aplicações gerais do sistema que não sejam de tempo real. Os principais modelos para esta política são: **SCHED_OTHER**, **SCHED_BAT** e **SCHED_IDLE**.

No contexto de tais políticas, o modelo SCHED_OTHER (ou *Time-Sharing* - TS) favorece um ambiente em que há um grande número de *threads* concorrendo por I/O (rede e/ou disco) e utilização de CPU. Este será o método utilizado pelas configurações do sistema para o Apache Hadoop e também é o método padrão do kernel do Linux (DOMINGO; BAILEY, 2016).

## 6.4 Grupos de Controles

Nos kernels atuais (a partir do kernel 2.6.24), um recurso conhecido como grupos de controle (ou cgroups) permitem alocar recursos como tempo da CPU (*time slice*), memória, largura de banda de rede ou combinações destes, entre grupos de tarefas (processos) em execução.

Neste trabalho utilizamos os grupos de controle para as tarefas do Java. Com a utilização de grupos de controle obtivemos redução considerável na latência entre as *threads* do Java em comparação com o Modelo Comum (de 21 ms para 3 ms). Uma análise mais detalhada é apresentada no Capítulo 8, com os resultados obtidos.

Esse é um dos pontos de contribuição deste trabalho. Além dos parâmetros de configuração do CFS que foram analisados, também logramos êxito com outras frentes de análise considerando o subsistema de tarefas em sua plenitude.

## 6.5 Comportamento do Escalonador e Parâmetros de Configurações

Em nossa análise dos principais parâmetros de configuração identificamos 14 (quatorze) parâmetros relacionados diretamente com o escalonador CFS. A partir desses parâmetros de configurações encontrados, identificamos que três parâmetros não eram relevantes pelo fato de tratarem de políticas para tarefas de tempo real.

Durante a análise e experimentos, os parâmetros de configurações consideramos relevantes, no sentido de afetar o desempenho do sistema, foram sched_child_runs_first, sched_migration_cost_ns, sched_latency_ns, sched_min_granularity_ns, sched_wakeup_granularity_ns, sched_rt_runtime_us e sched_nr_migrate.

Para a análise destes parâmetros de configurações utilizamos o utilitário **perf** nos experimentos realizados.

### 6.5.1 *sched_child_runs_first*

Um processo filho recém-bifurcado (criado) é executado antes que o processo pai continue a execução. Definir este parâmetro como 1 beneficia uma aplicação na qual o processo filho executa uma execução após o *fork*.

Em nossas análises, observamos que este parâmetro, se ativado, não possui efeito quando a tarefa utiliza a chamada de sistema vfork(), a qual é seguida por uma chamada de sistema exec()(STUART, 2010). Também, encontramos registros de anomalias no interpretador de comandos *bash* no Red Hat Enterprise Linux 6.

Apesar de este parâmetro de configuração não ter sido habilitado, consideramos que a ativação deste parâmetro de configuração pode trazer impactos negativos para o sistema.

### 6.5.2 ***sched_migration_cost_ns***

Parâmetro de configuração com grandeza em nanossegundos. Representa a quantidade de tempo após a última execução em que uma tarefa é considerada *"cache hot"* nas decisões de migração. Há uma baixa tendência que uma tarefa *"hot"* seja migrada, portanto, obervamos que aumentar essa variável reduz as migrações de tarefas entre os núcleos.

Este parâmetro de configuração, no Modelo Comum, possui o valor atribuído de 500000 (em nanossegundos). A partir de nossas análises, inferimos que de fato aumentar este valor teríamos dois impactos: favoreceríamos a estruturas de cache construídas, uma vez que a tarefa estaria vinculada à CPU em execução por mais tempo e também poderíamos minimizar o *overhead* da troca de contexto.

Em nossa análise do referencial teórico encontramos valores recomendados como 2500000 e 5000000. Com os experimentos obtivemos resultados melhores com valor 5000000. Estes resultados foram percebidos principalmente quanto à redução da migração de CPU, como apresentado na figura 7.

Neste sentido, havendo uma redução na migração de CPU, também obtivemos uma redução na troca de contexto, como observado na figura 8. Por outro lado, percebemos durante os experimentos que ao diminuir o valor padrão para 200000, tanto a migração de CPU quanto a troca de contexto aumentavam.

Realizamos um conjunto de cinco execuções com o WordCount para os valores 2000000, 2500000 e 5000000. Nestas execuções com o WordCount, o sistema operacional não foi reiniciado. O que inferimos é que este parâmetro em particular favorece a construção das estruturas de cache do processador, minimiza a migração de CPU e a troca de contexto.

A partir do referencial teórico e análises pertinentes, também observamos que se o tempo de inatividade da CPU for maior do que o tempo de espera quando houver processos executando, a redução do valor deste parâmetro de configuração deve ser considerada. Se as tarefas saltam entre as CPUs com muita frequência, aumentar este valor mitiga a migração de CPU e a troca de contexto.

Serviços como o Apache HTTP Server ou aplicações *multi-threaded* são favorecidos com o aumento deste valor. Em documentações oficiais da Red Hat tem-se o limite deste aumento em no máximo dez vezes o valor padrão. Esta métrica foi utilizado para limitar o número de experimentos, uma vez que conhecíamos o limite a ser atingido.

### 6.5.3 ***sched_latency_ns***

Parâmetro de configuração com grandeza em nanossegundos. Representa a latência da preempção direcionada para tarefas *CPU-bound*. Aumentar o valor deste parâmetro aumenta o tempo limite de uma tarefa *CPU-bound*. O peso da tarefa depende do nível de *nice* da tarefa e da política de escalonamento. O peso mínimo para uma tarefa SCHED_OTHER é 15, correspondente a *nice* 19. O peso máximo da tarefa é 88761, correspondente a *nice* -20.

Este parâmetro de configuração, fundamentalmente, aumenta o *time slice* das tarefas. O *time slice* corresponde ao valor de sched_latency_ns, porém, se o sistema possui um número de tarefas em fila para serem executadas e que esta fila excede a razão $\frac{sched\_latency\_ns}{sched\_min\_granularity\_ns}$, então o *time slice* é definido pelo produto do número de tarefas e sched_min_granularity_ns.

Este parâmetro de configuração, no Modelo Comum, possui o valor 60000000. Aumentar esse valor implica em também minimizar a troca de contexto, pois um dos parâmetros do kernel para realizar a troca de contexto é o *time slice* da tarefa terminar.

Neste sentido, optamos por minimizar a troca de contexto aumentando o *time slice*. Realizamos experimentos com os valores 70000000, 100000000 e 140000000. Nestes experimentos a máquina virtual não foi reiniciada. Foram realizados um conjunto de cinco execuções para cada valor.

O que percebemos é que a troca de contexto foi reduzida e conseguimos manter a latência entre as *threads* com 3ms utilizando o valor 100000000. Com os valores 120000000, 140000000 e 150000000 obtivemos um aumento na latência entre as *threads* para 8 ms, 11 ms e 14 ms, respectivamente (variações de 1 ms e 2 ms para mais e para menos ocorriam durante as duas primeiras execuções). A nossa conclusão é que se o número de *threads* escalar acima de 150 *threads* (como em nossos experimentos), aumentar o *time slice* aumenta a latência entre as *threads*.

### 6.5.4 ***sched_min_granularity_ns***

Parâmetro de configuração com grandeza em nanossegundos. Granularidade de preempção mínima para tarefas *CPU-bound*. Observamos que esse parâmetro depende do valor atribuído a sched_latency_ns. O kernel ajusta este valor com a seguinte equação sched_min_granularity_ns $= \frac{sched\_latency\_ns}{numero\_de\_tarefas}$, em que numero_de_tarefas corresponde ao número de tarefas na fila (KRISHNAMURTHY; ROUSKAS, 2015).

Caso a equidade não seja satisfeita o kernel aumenta o *time slice* para corresponder ao número de tarefas. O que observamos é que aumentar este valor (a partir do valor padrão, 750000), o aumento da latência entre as *threads* também aumenta. Neste sentido optamos por diminuir este valor. O critério inicial para esta decisão foi a partir do referencial teórico (DOMINGO; BAILEY, 2016) e trabalhos relacionados (KRISHNAMURTHY; ROUSKAS, 2015), que sugerem valores como 250000 e 100000.

Com este parâmetro de configuração realizamos cinco execuções durante os experimentos (a máquina virtual não foi reiniciada) para cada valor analisado. Os valores dos parâmetros de configurações anteriores (sched_latency_ns e sched_migration_cost_ns) foram mantidos com os valores ajustados.

Os valores analisados foram 50000, 100000 e 250000. O que observamos é que a latência entre as tarefas foram 11 ms, 7 ms e 8 ms, respectivamente.

Concluímos pelos experimentos realizados e como também já obervado (KRISHNAMURTHY; ROUSKAS, 2015) que o valor 100000 permite manter a latência reduzida com estes parâmetros. Com o valor 750000, do Modelo Comum (valor padrão do sistema), a latência entre as tarefas correspondia a 22 ms.

### 6.5.5 *sched_wakeup_granularity_ns*

Parâmetro de configuração com grandeza em nanossegundos. A granularidade de preempção de "acordar". Aumentar o valor deste parâmetro reduz a preempção de "acordar", reduzindo a "perturbação"das tarefas de *CPU-bound*. Este parâmetro, fundamentalmente, pode eliminar a preempção quando reduzido para zero (KRISHNAMURTHY; ROUSKAS, 2015).

Como métrica, primeiramente observamos tanto o referencial teórico (DOMINGO; BAILEY, 2016) quanto os trabalhos relacionados (KRISHNAMURTHY; ROUSKAS, 2015). Em nossos experimentos observamos que os valores 0 e 25000 possuiam resultados muitos próximos (sem variações abruptas na latência) entre 6 ms e 7 ms. Neste caso, para o Modelo Ajustado, consideramos tanto o valor 0 quanto o valor 25000.

### 6.5.6 *sched_nr_migrate*

Controla quantas tarefas podem ser movidas entre processadores por meio de interrupção de software de migração (*softirq*). Se um grande número de tarefas são criadas pela diretiva SCHED_OTHER, todas elas serão executadas no mesmo processador. O valor padrão é 32. Aumentar esse valor fornece um aumento de desempenho para muitas *threads* SCHED_OTHER em detrimento do aumento da latência para tarefas em tempo real.

Com a afinidade de processador realizada, observamos que aumentar este valor de fato reduz a troca de contexto, minimiza a migração de CPU e faz com que mais tarefas sejam executadas no núcleo em que a chamada de sitema clone() ou vfork() ocorreram.

Em nossos experimentos também observamos que atribuir os valores 256, 512 e 1024 impõe um aumento na utilização do processador. Considerando que o valor padrão é 32, optamos por realizar experimentos utilizando seus múltiplos. Os valores analisados foram 64, 128, 256, 512 e 1024. Este parâmetro influenciou principalmente no aumento da migração de CPU quando os valores 256, 512 e 1024 foram utilizados. Este fenômeno ocorreu pelo fato de a migração das tarefas para núcelos em estado de *idle* serem balanceadas, uma vez que um grande número de tarefas no mesmo núcleo impõe maior carga de utilização de CPU.

Concluímos, a partir dos nossos experimentos que os valores 64 e/ou 128 favoreceram a redução da migração de CPU, redução da troca de contexto e melhoram o tempo de execução.

Os resultados gerais aplicando estes parâmetros de configurações em conjunto e com os valores em que obtivemos os melhores resultados estão descritos no capítulo 9.

# 7 SUBSISTEMA DE MEMORIA VIRTUAL

O Linux, quanto ao subsistema de gerenciamento de memória virtual, possui dois componentes principais (SILBERSCHATZ; GALVIN; GAGNE, 2015):

- Alocação e liberação de memória física.
- Manipulação da memória virtual.

Sendo assim, o Linux oferece parâmetros de configurações para manipular determinadas características da alocação de memória. Neste trabalho consideramos duas perspectivas: **(i)** a alocação de memória e suas implicações para as tarefas do sistema e **(ii)** a alocação de memória quanto aos *buffers* de I/O de disco (abordado no próximo capítulo).

Consideramos os seguintes aspectos relevantes quanto ao gerenciamento de memória no âmbito de aplicações de missão crítica:

- Buscamos garantir que a propensão da utilização de memória de troca (*swap*) seja minimizada. Isto porque o acesso à memória secundária (disco rígido) é da ordem de milissegundos (5 - 10ms) enquanto que o acesso à memória RAM é da ordem de nanossegundos (50 - 100ns) (PATTERSON; HENNESSY, 2014). O que fica patente é que a utilização excessiva da memória de troca pode implicar em *overhead* devido ao tempo de acesso à memória secundária.
- Analisamos a melhor forma com a qual o Linux trata a reclamação de memória e os parâmetros de configurações correlatos.
- Analisamos a relação do subsistema de memória virtual com o subsistema de arquivos e I/O de disco e os parâmetros de configurações que se relacionam com essa abordagem.
- Garantir que em caso de escassez de memória devido a alta carga de trabalho do Apache Hadoop e aplicações de missão crítica, em hipótese alguma, sejam interrompidos pelo kernel.

Há aproximadamente 40 (quarenta) parâmetros relacionados à memória virtual, destes, uma parcela consiste de parâmetros somente para leitura, permitindo obter informações gerais do sistema, uma parcela trata de memória compartilhada e páginas grandes e outra parcela trata de questões da administração do sistema, segundo a documentação oficial em (RIEL; MORREALE, 2008).

O Linux utiliza-se de três componentes macros para a gestão do subsistema de memória virtual. São estes o *Zone Buddy Allocator* (alocação de páginas para todo o sistema), o *Slab Allocator* (alocação de tamanhos de páginas dinâmicas superiores ou inferiores a 4KB e monitoramento de páginas livres e alocadas) e por fim, as *threads* de kernel que auxiliam no gerenciamento da memória virtual. As *threads* que compõem o gerenciamento de memória são **kscand**, **kswapd**, **kupdated** e **pdflush** (ALVES, 2009).

## 7.1 Pontos Centrais do Subsistema de Memória Virtual e Parâmetros de Configurações

As tarefas do sistema, em sua maioria, requerem apenas páginas mapeadas para determinados endereços de virtuais. Porém, o kernel exige páginas de contíguas na memória física. Assim, o mecanismo de alocação de páginas comporta-se como um alocador de memória de tamanho variável, porém, utilizando o algoritmo do sistema *buddy* (STUART, 2010).

O alocador de páginas do sistema *buddy* suporta solicitações de páginas por zonas. O código solicitante utiliza-se a *flag* __GFP_DMA para a ZONE_DMA e a *flag* __GFP_HIGHMEM para a ZONE_HIGHMEM. Em caso de falhas de alocação de páginas na ZONE_DMA e ZONE_HIGHMEM, a solicitação é atendida pela ZONE_NORMAL. Em caso de não haver páginas livres na ZONE_NORMAL, então realiza-se o *swap* de páginas para o disco criando páginas livres suficientes para satisfazer a solicitação de página (STUART, 2010).

Para atender a solicitações de tamanhos de páginas não homogêneas o Linux utiliza-se do alocador *slab*. *Slabs* são coleções de blocos livres de memória de determinado tamanho (STUART, 2010).

### 7.1.1 *Tabela de Página*

A memória física é dividida em quadros de páginas e o Linux utiliza um modelo de tabela de página de quatro níveis para abstração. Os bits superiores do endereço virtual é utilizado para indexar o **diretório global de página - pgd** (*page global directory*). Quando uma entrada é selecionada, esta é um ponteiro para o **diretório superior de página - pud** (*page up directory*), o qual é indexado pelos bits mais significativos.

A entrada do pud, então, é um ponteiro para um **diretório médio de página - pmd** (*mid-level directory*). Por fim, o pmd aponta para a **tabela de página - pt** (*page table*). Os bits menos significativos são utilizados para deslocamento (*offset*).

Na arquitetura de 32 bits, tanto o **pgd** quanto o **pt** são indexados utilizando 10 bits do endereço virtual. Com os 12 bits de deslocamento, resulta-se em um tamanho de página de 4096 bytes. A política de substituição de páginas no Linux utilizam uma estrutura de dados referentes à política de **menos utilizada recentemente** - LRU (*Least Recently Used*), porém, a política de fato utilizada é uma combinação do algoritmo de "segunda chance"em conjunto com a política de "listas múltiplas"(STUART, 2010).

As listas de páginas são constituídas similarmente à técnica LRU 2Q, e também utiliza uma abordagem **não utilizada recentemente** - NRU (*Not Recently Used* para tratar páginas limpas ou inalteradas (*clean*) e páginas sujas ou modificadas (*dirty*).

### 7.1.2 *Estruturas de Buffer*

No contexto de estruturas de *buffer* no subsistema de memória virtual consideramos como relevantes as estruturas ***Page Cache*** e ***Buffer Cache***, ***Buffer Heads*** e ***Writeback***.

#### *Page Cache e Buffer Cache*

O *Page Cache* consiste de uma estrutura que representa o cache de dados. Todo dado que é lido do disco tem o conteúdo armazenado em uma entrada de *Page Cache* (GORMAN, 2009). Dados atualizados na Page Cache não requerem I/O de disco, caso tal dado seja requisitado. Quando uma página tem dados novos que não foram gravados, este dado é chamado de **sujo ou modificado** (*dirty*), ou seja, dados em *buffer* do *Page Cache*.

Páginas não classificadas como sujas são ditas **limpas ou inalteradas** (*clean*). Páginas de *Page Cache* limpas podem ser recuperadas em caso de falta de memória apenas se forem liberadas as páginas em cache, ou seja, *dirty pages*. As páginas classificadas como sujas devem primeiro ser limpas (escritas em disco) antes de serem recuperadas (ALVES, 2009).

O *Buffer Cache* é uma estrutura do *Page Cache* para dispositivos de bloco. Um sistema de arquivos normalmente usa o *Buffer Cache* ao acessar suas estruturas de metadados no disco, como tabelas de *inode* e mapas de bits de alocação.

Nos kernels atuais (a partir do kernel 2.6.32), o *Buffer Cache* e o *Page Cache* se constituem de uma única estrutura, a qual é denominada apenas como *Page Cache*.

#### Buffer Heads

Os *Buffer Heads* consistem de pequenas estruturas auxiliares que são alocadas no acesso ao *Page Cache* (GORMAN, 2009). Têm como vantagem a facilidade na recuperação de páginas limpas.

#### *Writeback*

Quando ocorre a escrita de dados em arquivos a Page Cache fica em estado de sujo. Quando as páginas estiverem neste estado ou quando a quantidade de memória suja atingir um número específico de páginas em bytes, o kernel inicia o processo de *writeback* (ALVES, 2009). As *threads* executam *writeback* em segundo plano e permitem que os aplicativos continuem executando.

## 7.2 Parâmetros de Configurações

Os tópicos citados possuem relação direta com determinados parâmetros de configurações do sistema. Os principais parâmetros que permitem alterar o comportamento do sistema quanto à memória virtual e que consideramos relevantes para este trabalho estão descritos a seguir (RIEL; MORREALE, 2008).

### 7.2.1 *Taxa de Reclamação de Memória*

Parâmetros que tratam de determinas características de reclamação de memória e principalmente *swap*, estão descritas na tabela 4.

Tabela 4 - Parâmetros pertinentes à reclamação de memória.

| **Parâmetro** | **Descrição** |
|---|---|
| vm.vfs_cache_pressure | Este parâmetro controla a tendência do kernel de recuperar a memória que é usada para o cache do VFS (*Virtual File System*). Aumentar esse valor aumenta a taxa na qual os caches do VFS são recuperados. |
| vm.swappiness | Esse parâmetro é usado para definir quão agressivamente o kernel troca a memória (swap) relativa a Page Cache e outros caches. Aumentar este valor aumenta a quantidade de operações de swap. |
| vm.min_free_kbytes | Controla a quantidade de memória que é mantida livre para uso extraordinário, incluindo alocações "atômicas"(aquelas que não podem esperar para serem recuperadas). O valor padrão depende da quantidade de memória física. |

### 7.2.2 *Writeback*

O processo de *writeback* consiste em escrever dados da *Page Cache* para o disco. A tabela 5 descreve os parâmetros que se relacionam com esse comportamento.

Tabela 5 - Parâmetros pertinentes à *writeback*.

| **Parâmetro** | **Descrição** |
|---|---|
| vm.dirty_background_bytes | Determina a quantidade de memória marcada como suja na qual a *thread* de kernel iniciará o processo de *writeback*. |
| vm.dirty_background_ratio | Representa a porcentagem total da quantidade de memória livre e recuperável. Quando a quantidade de *Page Cache* suja excede essa porcentagem, as *threads* de *writeback* iniciam a gravação de memória suja. |
| vm.dirty_bytes | Quantidade de memória suja para que uma tarefa inicie o *writeback*. O valor mínimo permitido para dirty_bytes são duas páginas (em bytes). Cada página possui 4096 bytes e valores inferiores são ignorados (arquitetura 32 bits). |
| vm.dirty_ratio | Valor percentual semelhante ao valor de dirty_background_ratio. Quando este é excedido, as aplicações que precisam gravar no *Page Cache* são bloqueadas e começam o processo de *writeback*. |

Há que se observar que os parâmetros da tabela 5 são mutuamente excludentes. Se um valor para vm.dirty_background_ratio ou vm.dirty_ratio é definido, então, valores definidos em vm.dirty_background_bytes e vm.dirty_bytes não são utilizados.

## 7.3 Alocação e Reclamação de Memória

Especificamente quanto à alocação de memória, dois parâmetros definem o comportamento do kernel:

**(i). vm.overcommit_memory**. Define como o Linux se comporta quando as reclamações de memória ocorrem. O valor padrão é 0 (zero), em que é aplicado um tratamento heurístico das reclamações de memória verificando se há memória disponível. O valor 1 (um) define que as reclamações de memória serão sempre atendidas, nesse caso, o kernel considera que sempre há memória disponível.

O valor 2 (dois), estabelece que as reclamações de memória só serão atendidas até o máximo de [swap + $\frac{overcommit\_ratio}{100}$ * RAMFísica] (GORMAN, 2009).

**(ii). vm.overcommit_ratio**. Representa o valor a ser utilizado quando o valor 2 (dois) é definido para o parâmetro de configuração vm.overcommit_memory.

Quando a memória do sistema começa a exaurir, o kernel deve tomar medidas para que a aplicação que está consumindo memória não gere uma interrupção geral no sistema. Esse fenômeno é conhecido como ***Out Of Memory Killer***, ou **OOM *Killer***. Nesse caso, alguma aplicação crítica pode ser interrompida, por exemplo, o Apache Hadoop (Java).

Cada processo recebe uma pontuação relacionada ao OOM, o oom_score. Esta pontuação distingue os processos mais propensos a serem interrompidos no sistema. Cada processo possui um arquivo chamado oom_adj. Esse parâmetro pode receber um valor entre -15 e +15. Quanto maior o valor, maior a chance de o processo ser escolhido para ser interrompido. Porém, caso o processo receba um valor de oom_adj de -17, o kernel entende que este processo não deve ser interrompido em situações de OOM.

## 7.4 Questões de Paginação

A paginação consiste na divisão da memória RAM em blocos de tamanhos fixos. Tais blocos são denominados páginas. Cada processo no sistema possui uma tabela de páginas para mapeamento das páginas utilizadas. Tipicamente, uma página é composta pelo endereço virtual, um bit de presença e um bit de acesso. O endereço virtual é tratado pela unidade de gerenciamento de memória (**MMU** – *Memory Management Unit*).

O bit de presença indica se a página está presente na memória principal e o bit de acesso indica se a página está sendo usada. As páginas são ditas livres *"free"* quando não estão alocadas e, portanto, está ativa *"active"* para ser alocada.

A *thread* de kernel, kscand, realiza uma varredura para determinar o estado de tais páginas e bem como modificar a variável ***count age*** a cada iteração, decrementado-a, até que seu valor chegue a zero e quando este valor é atingido a página será denominada inativa (*inactive*) (ALVES, 2009).

A página ativa apenas pode ser colocada como inativa no estado de sujo e nesse caso, ocorre o processo de ***page out***, em que a página é retirada da memória principal e movida para a memória secundária. Até o kernel 2.6.31 utilizava-se a *thread* ***pdflush*** (ALVES, 2009) para realizar este processo. A partir do kernel 2.6.32 é utilizado o *per-backing-device* baseado em *writeback*, que fornece *threads* para cada dispositivo de bloco.

### 7.4.1 *Tratamento de Páginas Sujas*

O parâmetro vm.dirty_background_ratio define quando as operações de *flush* (descarregamento) das páginas mapeadas como sujas irão ocorrer. O valor deste parâmetro corresponde ao percentual da memória física. Se a quantidade de bytes de páginas sujas atingirem esse valor, então a *thread pdflush* inicia o processo de enviar tais páginas para a memória secundária (ALVES, 2009).

Foi observado que se o valor definido em vm.dirty_background_ratio for baixo, por exemplo, abaixo de 10%, a frequência com que a *thread pdflush* executa é alta. A vantagem é que as páginas sujas diminuem; por outro lado, o acesso ao disco aumenta.

## 7.5 **Experimentos**

No subsistema de memória virtual não realizamos experimentos para comparações, somente para constatações e não foram apresentados neste trabalho. Os parâmetros de configurações neste subsistema possuem uma característica específica e unívoca. Neste sentido, optamos por minimizar a propensão a *swap*, alterar a heurística de reclamação de memória do kernel e minimizar a atuação das *threads* de kernel quanto ao *buffer* de memória. Os parâmetros de configurações no subsistema de memória virtual são pontuais e imediatos neste sentido.

# 8 SUBSISTEMA DE I/O DE DISCO E SISTEMA DE ARQUIVOS

Este capítulo aborda o subsistema de arquivos e I/O de disco. São pontuadas as questões relevantes quanto a estes subsistemas e analisados os seus principais parâmetros de configurações.

## 8.1 Subsistema de I/O de Disco

Os discos giram a uma velocidade dita **velocidade angular constante** (CAV – *Constant Angular Velocity*), porém, há um número maior de setores nas zonas mais externas que nas zonas mais internas, por outro lado, a utilização dos dados é eficiente em função da setorização multizona (SOMASUNDARAM; SHRIVASTAVA, 2011). A setorização multizona consiste no agrupamento das faixas em zonas com base na distância a partir do centro do disco. Neste sentido, as zonas mais externas (distantes do centro) possuem mais setores.

Dada tal característica, a taxa de transferência é reduzida se o acesso aos dados ocorrerem em zonas mais próximas do centro do *platter* (prato do disco) (SOMASUNDARAM; SHRIVASTAVA, 2011). Portanto, a partição das aplicações de missão crítica devem ficar nas zonas mais externas do disco. No caso de virtualização essa premissa é desconsiderada, mas relevante para ambientes que não contemplam virtualização.

Um disco de 5.400 RPM (rotações por minuto) possui uma latência rotacional (tempo gasto para o *platter* girar e posicionar os dados sob a cabeça de leiura/gravação) de 5,5 ms, enquanto que um disco de 15.000 RPM possui uma latência rotacional de 2 ms (SOMASUNDARAM; SHRIVASTAVA, 2011). É patente, portanto, que discos com menor latência rotacional irão proporcionar melhor desempenho de I/O para aplicações de missão crítica.

Também, os discos podem ser arranjados por meio de RAID (*Redundant Array of Independent Drives*). O nível RAID-0 possui melhor desempenho no processamento da fila de I/O por oferecer operações em paralelo, porém, a falha em um dos discos implica na perda completa dos dados.

O nível RAID-1 oferece tolerância a falhas, porém, não há melhorias no desempenho. Para alinhar desempenho com considerável tolerância a falhas há o arranjo RAID-0+1. Para operações de I/O que envolvem majoritamente a leitura de dados, o nível RAID-5 é uma consideração relevante.

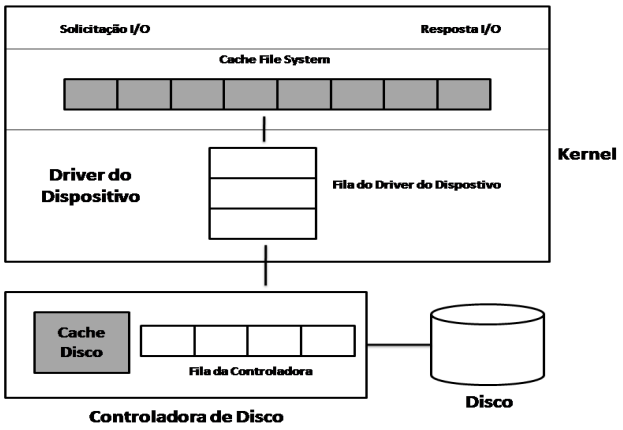
Apesar destes conceitos serem consolidados, há que se observar que se a aplicação de missão crítica requer intensivas operações de I/O de escritas e o grupo RAID na LUN (*Logical Unit Number*) do sistema de armazenamento for constituído de RAID-5, então o desempenho será afetado.

### 8.1.1 *Agendadores de I/O de Disco*

As aplicações ao realizarem as requisições de I/O acionam duas "classes" principais de um sistema de armazenamento: **Fila de I/O** e **Controlador de I/O** (SOMASUNDARAM; SHRIVASTAVA, 2011). A fila consiste de um *buffer* de solicitações de I/O que devem ser processadas pelo Controlador de I/O e este, por sua vez, processa tais requisições.

A figura 6 apresenta uma arquitetura típica de estrutura de armazenamento. A figura foi idealizada e retirada de (ALMEIDA; MENASCE, 2003), aqui foi adaptada para aderência com outras perspectivas.

Figura 6 - Estrutura do Subsistema de I/O de Disco.



No kernel do Linux a fila de I/O é tratada em um ambiente de processos concorrendo por I/O. O tratamento da fila é realizado por algoritmos do agendador de disco. Os principais agendadores de disco são o FCFS (*first-come, first-served*), SCAN (algoritmo do elevador), C-SCAN (circular SCAN) e LOOK (SILBERSCHATZ et al., 2015).

A partir da descrição dos principais agendadores de disco por (SILBERSCHATZ; GALVIN; GAGNE, 2015) e como (PRATT; HEGER, 2004) observa, tecnicamente o Linux não necessita de um agendador de I/O de disco implementado no kernel. Observaremos que o NOOP, agendador de I/O de disco no kernel do Linux, pode conduzir a melhor de desempenho caso o sistema de armazenamento subjacente seja de alto desempenho.

O Linux oferece os seguintes algoritmos de agendadores de I/O: **NOOP** (*No-operation*), **CFQ** (*Completely Fair Queuing*) e **Deadline**. O agendador de I/O Antecipatory não será abordado pelo fato de não estar disponível na versão de kernel utilizada neste trabalho.

### 8.1.2 *NOOP*

Dentre os agendadores, este é o agendador mais "simples" pelo fato de implementar o algoritmo baseado em FIFO (*first-in, first-out*) (REZGUI et al., 2014). O processo de *merging* é realizado na camada de bloco genérica. Este agendador oferece *overhead* mínimo ao sistema por oferecer somente mecanismos básicos de mesclagem e classificação.

O NOOP é uma opção considerável caso o sistema de armazenamento subjacente seja baseado em RAID, uma vez que o software da controladora requer pouca interação com o kernel por possuir seus próprios algoritmos de agendador de I/O (PRATT; HEGER, 2004).

### 8.1.3 *CFQ*

Este é o agendador padrão do Red Hat Enterprise Linux 6. Em sua essência, o agendador busca manter equidade e justiça quanto ao atendimento das requisições de I/O por meio de um grupo de filas (para cada tarefa) atendidas homogeneamente.

O CFQ possui três programações de classes: **tempo-real** (RT – *Real-Time*), **melhor-esforço** (BE – *Best-Effort*) e **ocioso** (*idle*). As classes são manipuladas no espaço do usuário pelo utilitário ionice e no espaço do kernel por meio da chamada de sistema ioprio_set.

A classe padrão para os processos do sistema é a classe melhor-esforço. Tanto a classe melhor-esforço quanto a classe tempo-real possuem oito subdivisões que definem as prioridades e estas variam entre 0 e 7, sendo 0 a prioridade mais alta.

### 8.1.4 *Deadline*

Este é o agendador padrão do Ubuntu Server 14.04 e busca garantir uma determinada latência para as solicitações de escrita. Neste agendador, por padrão, as leituras recebem prioridade sobre as operações de escrita.

Como observado em (PRATT; HEGER, 2004), o princípio do agendador Deadline (prazo) é que todas as solicitações de leitura sejam atendidas dentro de um período (prazo) de tempo especificado. Por outro lado, os pedidos de escrita não têm prazos específicos associados.

### 8.1.5 *Sistema de Arquivos*

O sistema de arquivos é o aspecto mais reconhecível de um sistema operacional (STUART, 2010). Neste sentido, o Linux utiliza uma divisão de trabalho com dois níveis de abstração. No nível superior há o **sistema de arquivos virtual** (VFS - *Virtual File System*) que oferece uma interface (suporte e semântica a serviços comuns e operações gerais) para os **sistemas de arquivos**, em um nível inferior.

Com a conceitualização do VFS o Linux é possibilitado a implementar diversos sistemas de arquivos, entre eles o MinixFS, Ext2, Ext3, Ext4, RaiserFS e ZFS. Neste trabalho foi contemplado o sistema de arquivos Ext4.

O Ext4 possui um tamanho de bloco padrão de 4096 bytes. O tamanho do bloco possui uma relação de otimização que depende do tamanho (em bytes) dos arquivos manipulados pela aplicação. (SOMASUNDARAM; SHRIVASTAVA, 2011), observou as seguintes particularidades quanto ao tamanho do bloco dispostas na tabela 6:

Tabela 6 - Relação do Tamanho de Bloco, Tempo de Transferência e IOPS.

| **Block Size** | **Tempo Transferência** | **IOPS** |
|---|---|---|
| 4 KB | 4 KB / 160 MB = 0.025 | 1 / (0.3 + 0.025) = 3,076 |
| 8 KB | 8 KB / 160 MB = 0.05 | 1 / (0.3 + 0.05) = 2,857 |
| 16 KB | 16 KB / 160 MB = 0.1 | 1/ (0.3 + 0.1) = 2,500 |
| 32 KB | 64 KB / 160 MB = 0.4 | 1 / (0.3 + 0.2) = 2,000 |
| 64 KB | 32 KB / 160 MB = 0.2 | 1 / (0.3 + 0.4) = 1,428 |

Esta análise considerou o tempo de serviço, ou seja, o período de tempo despendido no qual uma solicitação está recebendo o serviço de um determinado recurso (ALMEIDA; MENASCE, 2003), no caso, o disco, como 0,3 ms e a taxa de transferência de 160 MB/s. Nessas condições, o tempo de transferência de um bloco de 64 KB é maior que o tempo de transferência de um tamanho de bloco de 4 KB, padrão do Ext4 (4096 bytes). Porém, em blocos de 64 KB, as operações de I/O (IOPS) são menores.

A partir de tal conjectura, o tamanho do bloco não será alterado pelo fato de que os resultados seriam condicionados a um universo menor no que tange às possibilidades dos parâmetros de configurações, restringindo parte dos resultados obtidos a uma característica muito específica e peculiar da carga de dados.

A análise de (SOMASUNDARAM; SHRIVASTAVA, 2011) é suficiente para lançar luz de que o tamanho do bloco, em função do tamanho dos dados unitários manipulados pela aplicação é dependente do que se almeja como parâmetro: tempo de transferência menor ou operações de I/O menores. Isto é uma particularidade que não será tratada neste trabalho, mas possui relação direta com o sistema de arquivos e objeto de estudo.

### 8.1.6 *Opções do Sistema de Arquivos Ext4*

O sistema de arquivos Ext4 possui um conjunto de parâmetros que permitem alterar seu comportamento no tratamento de dados. Estes parâmetros se relacionam com o registros de informações pertinentes ao **tempo do acesso aos arquivos** (criação, modificação, alteração) e com a **política de journaling**.

Quando um arquivo é lido, o tempo de acesso (atime) para esse arquivo é atualizado no *inode*, o que requer uma operação de I/O adicional de escrita. A diretiva de tempo relativo (relatime), disponível no Ext4, mitiga a sobrecarga pelo fato de atualizar o tempo de acesso somente se o tempo de acesso anterior for mais antigo do que o tempo de modificação (mtime) ou o tempo de alteração de status (ctime). Na partição do Apache Hadoop, tanto noatime quanto nodiratime (processo semelhante a atime, porém, para diretórios), foram **habilitados**.

Também, outra questão envolve o mecanismo de **barrier**. O *barrier* de escrita é um mecanismo do kernel para garantir que os metadados do sistema de arquivos sejam escritos e ordenados corretamente mesmo em queda de energia. Este mecanismo é habilitado por padrão. O sistema de armazenamento deve suportar esse mecanismo. A opção de *barrier* foi desativada no ambiente de experimentos.

Quanto à política de journaling, três diretivas são relevantes para esta análise: **Journal**, **Ordered** e **Writeback**. Estas políticas implicam em duas perspectivas: **integridade de dados** e **desempenho**.

A política de *Journal* oferece melhor nível de integridade aos dados ao custo de um *overhead* maior por estender a integridade para a seção de dados do superbloco (não somente aos metadados). Tanto a política *Ordered* (padrão do sistema) quanto *Writeback* oferecem integridade somente na seção de metadados do superbloco, o que difere essas duas políticas é quanto à ordem de gravação.

A política *Ordered* faz com que os dados sejam gravados antes dos metadados serem atualizados na área de *journaling*. A política *Writeback* realiza somente a atualização dos metadados e nenhuma operação é ralizada sobre os dados dos arquivos.

## 8.2 Parâmetros de Configurações do Subsistema de I/O de Disco

Em nossas análises consideramos como relevantes para este trabalho, um conjunto de cinco parâmetros de configurações, os quais afetam o desempenho do subsistema de I/O de disco. Particularmente, observamos que os parâmetros nr_requests e read_ahead_kb são os principais influenciadores no desempenho.

### 8.2.1 *max_sectors_kb*

O valor mínimo é limitado pelo tamanho do bloco lógico e o valor máximo é limitado por max_hw_sectors_kb. Observamos que esta variável afeta principalmente discos de estado sólido (SSD).

O tamanho do bloco lógico pode ser obtido pela variável logical_block_size (512 bytes, no ambiente de experimentos). A variável max_hw_sectors_kb possui o valor de 4096 (bytes).

Análises exclusivas com alterações neste parâmetro de configuração (max_sectors_kb) com os valores 4096, 8192 e 1280, não produziram melhorias significativas no *throughput*. Também, não realizamos análises em discos de estado sólido (SSD).

### 8.2.2 ***nomerges***

Este parâmetro auxilia no processo de depuração. A maioria das cargas de trabalho se beneficia da fusão de solicitações (mesmo em armazenamento mais rápido, como SSD).

Para este parâmetro de configuração não alteramos o valor padrão.

### 8.2.3 ***nr_requests***

Cada fila de solicitações tem um limite no número total de descritores de solicitações que podem ser alocados para cada operação de I/O de leitura e escrita. Por padrão, o número é 128, o que significa que 128 requisições de leituras e 128 requisições de escritas podem ser enfileiradas antes de colocar uma determinada tarefa para dormir.

Uma vez que nr_requests tenha sido atribuído, todos os outros processos que tentam realizar operações de I/O serão colocados para dormir e esperar até que as solicitações estejam disponíveis.

Observamos que este parâmetro de configuração influencia muito no *throughput*. Realizamos experimentos com o TestDFSIO e o TIOBench com os valores 64, 128, 256, 512, 1024, 2048 e 4096, com variações no número de *threads* e tamanhos e quantidades de arquivos para os agendadores de I/O de disco NOOP, Deadline e CFQ. Mais informações estão dispostas no capítulo 9.

Em nossos experimentos consideramos as escolhas dos valores 64, 128, 256, 512, 1024, 2048 e 4096 baseado em nossas análises do referencial teórico. Também, estes valores são múltiplos.

### 8.2.4 ***read_ahead_kb***

O kernel do Linux pode detectar quando uma aplicação está lendo dados sequencialmente. Nesses casos, o kernel executa um algoritmo inteligente de leitura antecipada (*read-ahead*), em que mais dados do que os solicitados pelo usuário são lidos a partir do disco. Sendo assim, quando a tarefa tenta ler um bloco de dados adjacente, ele já estará no cache de página do kernel.

Por outro lado, o kernel pode ler mais dados do disco do que o necessário, o que ocupa espaço no cache de páginas até ser despejado por causa da alta pressão da memória. Para dispositivos RAID e LVM, geralmente aumentar o valor de read_ahead_kb para um número grande, como 8192, pode produzir melhor desempenho.

Observamos também que este parâmetro de configuração é relevante quanto ao *throughput*. Em nossos experimentos não consideramos a execução unívoca deste parâmetro de configuração. Consideramos alinhar este parâmetro de configuração com o parâmetro nr_requests. Adotamos esta abordagem pelo fato de entendermos, a partir do referencial teórico, que estes são complementares.

Nos experimentos, considerando um ambiente de virtualização com um sistema de armazenamento em rede, somente ajustamos o valor para o dobro do valor padrão (128), portanto, 256. É possível que para discos locais com sistema de RAID ou LVM este parâmetro de configuração obtenha melhores resultados.

Em nossos experimentos, considerando um ambiente de virtualização, utilizamos o TIOBench para gerar leitura de dados sequenciais, porém, não observamos melhorias significativas no *throughput* para os valores 2048, 4096 e 8192. Especificamente quanto à este parâmetro de configuração, realizamos três execuções do TIOBench para leituras sequenciais com os valores 2048, 4096 e 8192. A máquina virtual foi reiniciada em cada experimento.

Escolhemos estes valores por serem múltiplos e um dos valores recomendados pela documentação oficial do kernel do linux (KERNEL.ORG, 2017), sugerir o valor 8192.

### 8.2.5 *rq_affinity*

As respostas de I/O podem ser processadas em uma CPU diferente daquela que emitiu a requisição de I/O. A definição de rq_affinity para 1 faz com que o kernel forneça as respostas das solicitações de I/O à CPU na qual a requisição de I/O foi emitida. Isso pode melhorar a eficiência do cache de dados da CPU.

O valor deste parâmetro de configuração não foi alterado em nossos experimentos. Ainda assim, observamos a partir do referencial teórico que o valor deste parâmetro de configuração deve ser mantido na maioria dos ambientes (DOMINGO; BAILEY, 2016).

# 9 ANÁLISES E RESULTADOS OBTIDOS

Este capítulo aborda os resultados obtidos durante os experimentos realizados. São apresentados os resultados particulares de cada subsistema e uma análise consolidada é apresentada considerando a implementação de todos os parâmetros de configurações.

## 9.1 Análise e Resultados Obtidos - Subsistema de Tarefas

Inicialmente, iremos abordar as análises pertinentes e os resultados obtidos a partir dos experimentos, o subsistema de tarefas. A escolha do subsistema de tarefas é a primeira etapa da metodologia utilizada neste trabalho. As etapas da metodologia utilizada encontram-se na figura 4.

Seguindo a metodologia, a primeira etapa foi definida com a escolha do subsistema de tarefas. A etapa seguinte, portanto, envolveu a realização de experimentos aplicando a carga de trabalho para análises posteriores.

Nestes experimentos nós utilizamos o WordCount, um código desenvolvido para contar palavras de um determinado arquivo. O arquivo é lido em disco e processado.

O maior arquivo de texto utilizado durante as realizações dos experimentos foi de 1.1 GB e continha 181.558.984 palavras. Foram realizadas o mínimo de cinco execuções e para cada execução a máquina virtual foi reiniciada. Neste sentido, optamos por apresentar os parâmetros de configurações, que quando ajustados com um determinado valor a partir dos experimentos realizados, produziram melhor desempenho que no Modelo Comum, ou seja, com os parâmetros de configurações nativos do kernel do Linux.

### 9.1.1 *Análise Inicial do Ambiente de Experimentos*

Nos experimentos realizados observamos as seguintes características durante as execuções do código do WordCount:

- Não foram realizados ajustes de desempenho no HDFS ou JVM;
- Foram iniciados quatro processos principais do Java;
- O número de *threads* iniciadas pelo processo Java referente ao *Data Node* foram 89 (oitenta e nove) *threads* e o número de *threads* iniciadas pelo processo Java para o código do WordCount foram 187 (cento e oitenta e sete) *threads*. Outras *threads* auxiliares do Java eram de aproximadamente 60 (sessenta) *threads*.
- Quanto à execução do código do WordCount, os processos e suas respectivas *threads* estiveram suas execuções realizadas em todos os núcleos da CPU, havendo um consumo de CPU total oscilando entre 100% a 110% distribuídos entre os núcleos (esta informação foi obtida com o comando **top**).

### 9.1.2 *Análises Pertinentes ao Modelo Comum*

Primeiramente, como na metodologia utilizada, nós iniciamos as etapas de carga de trabalho, coleta de dados e análise no Modelo Comum, com os valores dos parâmetros de configurações nativos do kernel:

- Nenhuma afinidade de processador foi realizada, ou seja, todos os processos e *threads* do Java utilizaram todos os processadores disponíveis.
- O número de arquivos abertos estava definido como 1024.

Os valores dos parâmetros de configurações nativos do sistema (Modelo Comum) estão descritos na tabela 7. Estes parâmetros de configurações foram catalogados por meio do utilitário **sysctl**. As referências do referêncial teórico também apresentam estes parâmetros de configurações:

Tabela 7 - Parâmetros de Configurações do Modelo Comum.

| **Parâmetro** | **Configuração** | **Descrição** |
|---|---|---|
| sched_migration_cost_ns | 5000000 | Quantidade de tempo após a última execução em que uma tarefa é considerada *"cache hot"* nas decisões de migração. |
| sched_autogroup_enabled | 1 | Habilita a utilização de grupos para gestão de recursos. |
| sched_nr_migrate | 32 | Quantidade de tarefas que podem ser movidas entre os processadores. |
| sched_wakeup_granularity_ns | 1000000 | A granularidade de preempção de "acordar". |
| sched_min_granularity_ns | 750000 | Granularidade de preempção mínima para tarefas *CPU-bound*. |
| sched_latency_ns | 6000000 | Latência da preempção direcionada para tarefas *CPU-bound*. |
| fs.file-max | 254462 | Número máximo de arquivos manipulados por todo o sistema. |
| nofile | 1024 | Número máximo de arquivos que podem ser abertos no sistema por usuário ou por grupo de usuários. |

### 9.1.3 *Modelo Ajustado – Subsistema de Gerenciamento de Tarefas*

Nesse momento, iremos descrever o ambiente de experimentos contemplando as seguintes características quanto ao Modelo Ajustado, ou seja, com as configurações analisadas, testadas e configuradas como parte da metodologia proposta.

Neste primeiro momento, estaremos apresentando tanto os ajustes gerais quanto os valores dos parâmetros de configurações, em que após o conjunto das cargas de trabalho nos experimentos realizados, obtivemos os melhores resultados:

- Realizamos a afinidade de processador para todos os processos e *threads* do Java (Apache Hadoop) nos núcleos 0, 1 e 2 (a máquina virtual em questão possui quatro núcleos).

- O número de arquivos abertos foi definido para 665.536. No Modelo Comum, com as configurações nativas do sistema operacional, este valor correspondia a 1024.

Os parâmetros de configurações para o Modelo Ajustado estão descritos na tabela 8. Estes valores correspondem aos melhores resultados que obtivemos durante as etapas 3, 4 e 5 da metodologia proposta quanto ao Modelo Ajustado, como pode ser observado na figura 4:

Tabela 8 - Parâmetros de Configurações do Modelo Ajustado.

| **Parâmetro** | **Valor** | **Descrição** |
|---|---|---|
| sched_migration_cost_ns | 50000000 | Quantidade de tempo após a última execução em que uma tarefa é considerada *"cache hot"* nas decisões de migração. Este valor representou o melhor desempenho com experimentos envolvendo os valores 2000000, 2500000 e 5000000. |
| sched_autogroup_enabled | 1 | Habilita a utilização de grupos para gestão de recursos. |
| sched_nr_migrate | 64 | Quantidade de tarefas que podem ser movidas entre os processadores. Este valor representou o melhor desempenho com experimentos envolvendo os valores 64, 128, 256, 512 e 1024.. |
| sched_rt_runtime_us | Não aplicável | Uma vez que foi adotado o escalonador SCHED_OTHER, essa variável não é configurada. |
| sched_wakeup_granularity_ns | 0/25000 | A granularidade de preempção de "acordar". Este valor representou o melhor desempenho com experimentos envolvendo os valores 0 e 25000 |
| sched_min_granularity_ns | 100000 | Granularidade de preempção mínima para tarefas *CPU-bound*. Este valor representou o melhor desempenho com experimentos envolvendo os valores 50000, 100000 e 250000. |
| sched_latency_ns | 1000000 | Latência da preempção direcionada para tarefas *CPU-bound*. Este valor representou o melhor desempenho com experimentos envolvendo os valores 70000000, 100000000 e 140000000. |
| fs.file-max | 665536 | Número máximo de arquivos manipulados pelo sistema. |
| nofile | 665536 | Número máximo de arquivos que podem ser abertos no sistema por usuário ou por grupo de usuários. |

### 9.1.4 *Resultados Obtidos no Modelo Comum*

A tabela 9 apresenta as características da entrada de dados quanto ao tamanho do arquivo e a quantidade de palavras em cada arquivo durante os experimentos. Este conjunto de arquivos foi utilizado como entrada para o código do WordCount:

Tabela 9 - Arquivos de entrada de dados para o Modelo Comum e Modelo Ajustado.

| **Arquivo** | **Tamanho** | **Palavras** |
|---|---|---|
| 4300-0.txt | 1.5 MB | 268.034 |
| 5000-8.txt | 1.4 MB | 252.095 |
| big.txt | 6.2 MB | 1.095.695 |
| count_1w.txt | 4.8 MB | 666.666 |
| count_2w.txt | 5.4 MB | 859.074 |
| pg132.txt | 336 KB | 57.336 |
| pg1661.txt | 581 KB | 107.533 |
| pg19699.txt | 1.9 MB | 317.823 |
| pg20417.txt | 659 KB | 109.844 |
| pg972.txt | 385 KB | 63.131 |
| shakespeare.txt | 4.4 MB | 980.637 |
| huge.txt | 1.1 GB | 181.558.984 |
| Total bytes/palavras | 1.111.482.021 | 186.336.852 |

As ferramentas do Linux utilizadas para suporte às análises das execuções foram os utilitários de sistema top, ps, perf, strace e time (/usr/bin/time). Parte destas ferramentas e suas respectivas versões estão listadas na tabela 1.

Na tabela 10 são apresentados os resultados gerais quanto ao tempo de execução do código do WordCount no Modelo Comum, ou seja, com os valores dos parâmetros de configurações nativos do kernel do Linux.

Durante os experimentos foram realizadas cinco execuções do código do WordCount para o Modelo Comum. O objetivo foi analisar os resultados, catalogando-os, para comparações elementares com o Modelo Ajustado.

Na tabela 10 optamos por apresentar os resultados de três execuções consecutivas pelo fato de os resultados das execuções quatro e cinco serem semelhantes aos resultados obtidos com as execuções 1, 2 e 3.

Tabela 10 - Execução do WordCount para o Modelo Comum.

| **Execução** | **Aplicação** | **Arquivos Entrada** | **Bytes Entrada** | **Palavras Analisadas** | **Tempo Execução** |
|---|---|---|---|---|---|
| 01 | WordCount | 12 | 1.111.482.021 | 186.336.852 | 12:53.92 |
| 02 | WordCount | 12 | 1.111.482.021 | 186.336.852 | 11:48.74 |
| 03 | WordCount | 12 | 1.111.482.021 | 186.336.852 | 11:34.70 |

Como complemento, utilizamos a tabela 11 para apresentar dados quanto à **troca de contexto** e utilização da CPU durante a execução do código do WordCount. As informações referentes à utilização da CPU foram obtidas por meio do utilitário **top**, que é nativo no sistema operacional Linux. Informações referentes à troca de contexto foram obtidas com o utilitário **perf**. O tempo de execução foi analisado com o utilitário **time**.

Tabela 11 - Dados complementares da execução do WordCount no Modelo Comum.

| Execução | Aplicação | Troca Contexto | CPU |
|---|---|---|---|
| 01 | WordCount | 39.271 | 100% |
| 02 | WordCount | 34.206 | 101% |
| 03 | WordCount | 47.914 | 99% |

Na tabela 12 apresentamos dados quanto à **migração de CPU**, **falhas de página (*page faults*)** e **instruções por ciclo** da CPU durante a execução do código do WordCount. As informações referentes à migração de CPU, falhas de página e instruções por ciclo da CPU foram obtidas com o utilitário **perf**.

Tabela 12 - Dados complementares da execução do WordCount no Modelo Comum quanto à migração de CPU, falhas de páginas e instruções por ciclo da CPU.

| Execução | Migração de CPU | Page Faults | Instruções p/ Ciclo |
|---|---|---|---|
| 01 | 2.714 | 32.462 | 2,11 |
| 02 | 2.799 | 32.347 | 2,11 |
| 03 | 2.777 | 32.497 | 2,06 |

### 9.1.5 *Resultados Obtidos no Modelo Ajustado*

No modelo ajustado realizamos cinco execuções para cada parâmetro de configuração ajustado disposto na tabela 8. Para cada execução do código do WordCount foi realizada a reinicialização da máquina virtual.

Os parâmetros fs.file-max e nofile não foram experimentados pelo fato de estes parâmetros de configurações seguirem as melhores práticas preconizadas (DOMINGO; BAILEY, 2016). Também, estes parâmetros de configurações foram considerados como um pré-requisito para aplicações de missão crítica.

As definições dos principais parâmetros do subsistema de tarefs e os critérios envolvidos para a seleção destes e os valores correspondentes estão descritos no capítulo 6. As ferramentas utilizadas para suporte às análises das execuções foram os utilitários de sistema top, ps, perf, strace e time (/usr/bin/time).

Tanto no Modelo Comum quanto no Modelo Ajustado, as tabelas comparativas dispostas neste capítulo apresentam os resultados obtidos com as execuções 3, 4 e 5 do código do WordCount. Também, as tabelas dispostas quanto ao Modelo Ajustado estão considerando todo o conjunto de experimentos, ou seja, após todos os experimentos realizados com cada parâmetro de configuração ajustado.

A análise considerando os experimentos com os parâmetros de configuração de forma individual tinha como objetivo primário compreender se um determinado parâmetro em particular influenciva no desempenho.

Não foram catalogados e analisados dados considerando parâmetros de configurações particulares, mas sim, um modelo (Modelo Comum), compreendendo um conjunto de parâmetros de configurações máximo que impactam no desempenho.

Sendo assim, considerando a nossa metodologia e objeto de estudo, privilegiamos um conjunto de parâmetros de configurações que representam um modelo, no caso, o Modelo Ajustado. Nos capítulos 6, 7 e 8, são apresentados os parâmetros de configurações relacionados a cada subsistema do kernel do Linux. Nestes capítulos estão descritos as motivações das escolhas dos parâmetros de configurações e bem como os valores ajustados.

Para este capítulo em particular, consideramos exclusivamente os resultados gerais, apresentado a conclusão dos experimentos realizados privilegiando a comparação entre os modelos propostos; Modelo Comum e Modelo Ajustado.

Na tabela 13 é apresentado os resultados gerais quanto ao tempo de execução do WordCount no Modelo Ajustado. Nesta tabela são apresentados os resultados considerando três execuções para fins comparativos com as execuções do código do WordCount com o Modelo Comum.

Tabela 13 - Execução do WordCount para o Modelo Ajustado.

| **Execução** | **Aplicação** | **Arquivos Entrada** | **Bytes Entrada** | **Palavras Analisadas** | **Execução** |
|---|---|---|---|---|---|
| 01 | WordCount | 12 | 1.111.482.021 | 186.336.852 | 10:03.79 |
| 02 | WordCount | 12 | 1.111.482.021 | 186.336.852 | 09:23.90 |
| 03 | WordCount | 12 | 1.111.482.021 | 186.336.852 | 08:47.54 |

Como complemento, utilizamos a tabela 14 para apresentar dados quanto à **troca de contexto** e utilização da CPU durante a execução do código do WordCount. As informações referentes à utilização da CPU foram obtidas por meio do utilitário **top**, que é nativo no sistema operacional Linux. Informações referentes à troca de contexto foram obtidas com o utilitário **perf**. O tempo de execução foi analisado com o utilitário **time**.

Tabela 14 - Dados complementares da execução do WordCount no Modelo Ajustado.

| **Execução** | **Aplicação** | **Troca Contexto** | **CPU** |
|---|---|---|---|
| 01 | WordCount | 35.954 | 101% |
| 02 | WordCount | 33.714 | 101% |
| 03 | WordCount | 33.156 | 101% |

Na tabela 15 apresentamos dados quanto à **migração de CPU**, **falhas de página** (***page faults***) e **instruções por ciclo** da CPU durante a execução do código do WordCount. As informações referentes à migração de CPU, falhas de página e instruções por ciclo da CPU foram obtidas com o utilitário **perf**.

Tabela 15 - Dados complementares da execução do WordCount no Modelo Ajustado quanto à migração de CPU, falhas de páginas e instruções por ciclo da CPU.

| **Execução** | **Migração de CPU** | **Page Faults** | **Instruções p/ Ciclo** |
|---|---|---|---|
| 01 | 2.693 | 32.002 | 2,12 |
| 02 | 2.411 | 32.298 | 2,10 |
| 03 | 2.280 | 34.050 | 2,12 |

### 9.1.6 *Análise dos Dados*

Iniciaremos a análise dos dados obtidos utilizando a tabela 16, a tabela 17 e tabela 18 apresentando os parâmetros mais relevantes na coleta de dados. Estas tabelas sintetizam de forma comparativa o tempo de execução, a troca de contexto, tempo dedicado ao sistema, migração de CPU, falha de páginas e instruções por ciclo.

Tabela 16 - Comparação da execução do código do WordCount no Modelo Comum e Modelo Ajustado considerando o tempo de execução e troca de contexto.

| **Modelo Comum** | | **Modelo Ajustado** | |
|---|---|---|---|
| **Execução** | **Troca Contexto** | **Execução** | **Troca Contexto** |
| 12:53.92 | 39.271 | 10:03.79 | 35.954 |
| 11:48.74 | 34.206 | 09:23.90 | 33.714 |
| 11:34.70 | 47.914 | 08:47.54 | 33.156 |

Tabela 17 - Comparação da execução do código do WordCount no Modelo Comum e Modelo Ajustado considerando a migração de CPU e falhas de página.

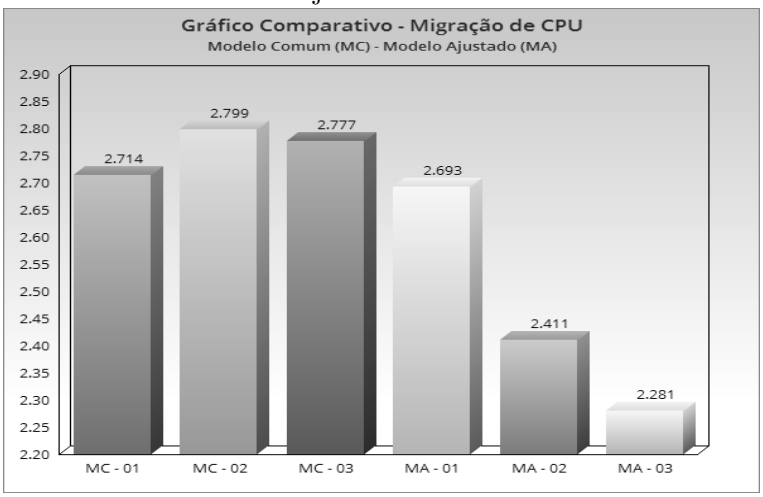
| **Modelo Comum** | | **Modelo Ajustado** | |
|---|---|---|---|
| **Migração CPU** | **Page Faults** | **Migração CPU** | **Page Faults** |
| 2.714 | 32.462 | 2.693 | 32.002 |
| 2.799 | 32.347 | 2.411 | 32.298 |
| 2.777 | 32.497 | 2.280 | 34.050 |

Tabela 18 - Comparação da execução do código do WordCount no Modelo Comum e Modelo Ajustado considerando as instruções por ciclo de CPU.

| **Modelo Comum** | **Modelo Ajustado** |
|---|---|
| **Instruções p/ Ciclo** | **Instruções p/ Ciclo** |
| 2,11 | 2,12 |
| 2,11 | 2,10 |
| 2,06 | 2,12 |

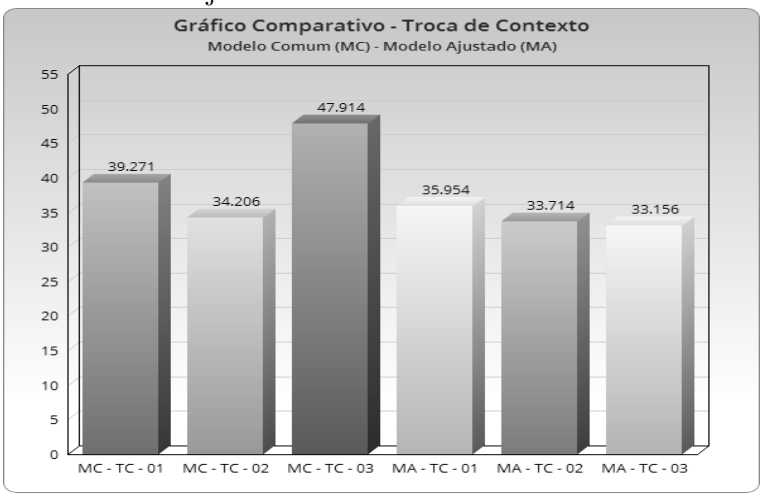
Também, utilizamos um gráfico para representar esses dados. Considerando o melhor caso do Modelo Comum e o melhor caso do Modelo Ajustado, na figura 7, observamos que a migração entre os núcleos foi reduzida de 2.714 (Modelo Comum) ocorrências de migração de CPU para 2.281 (Modelo Ajustado). Isto representa um fator de melhoria de 18.9%. A diminuição das migrações entre os núcleos podem reduzir o *overhead* da troca de contexto entre os núcleos.

Figura 7 - Comparativo da migração de CPU na execução do Apache Hadoop (WordCount) no Modelo Comum e Modelo Ajustado.




Quanto à **troca de contexto**, considerando o melhor caso do Modelo Comum e o melhor caso do Modelo Ajustado, na figura 8, observamos que a troca de contexto foi reduzida de 34.206 (Modelo Comum) ocorrências de troca de contexto para 33.156 (Modelo Ajustado). Isto representa um fator de melhoria de 3.16%.

Figura 8 - Comparativo da troca de contexto na execução do Apache Hadoop (wordcount) no Modelo Comum e Modelo Ajustado.




Outro aspecto importante é a **latência máxima** (em milissegundos) entre as *threads*. Essa latência foi reduzida de **21ms** do Modelo Comum para até **3ms** no Modelo Ajustado, ou seja, um fator de melhoria de 600%. O que contribuiu para esse ganho em potencial foi colocar os processos do Apache Hadoop no mesmo grupo de controle (*cgroups*), realizar a afinidade de processador e os parâmetros de configurações pertinentes ao escalonador CFS.

Observarmos as seguintes questões fundamentais que permitiram obter melhor desempenho no subsistema de tarefas:

(i). **Afinidade de CPU.** A afinidade de CPU foi realizada para um número menor de núcleos, porém, dedicados. Essa medida permite que a troca de contexto seja minimizada.

(ii). Parâmetro **sched_migration_cost_ns.** Este parâmetro foi modificado de 500000 ns para 5000000 ns. Este aumento no parâmetro de configuração faz com que as tarefas sejam tratadas de forma que a migração entre os núcleos da CPU seja adiada. Este valor foi obtido pela pesquisa realizada por meio do referencial teórico e por meio dos trabalhos relacionados (KRISHNAMURTHY; ROUSKAS, 2015). Experimentos também foram realizados o valor 2500000, que não produziu melhores resultados. Mais informações sobre este parâmetro de configuração e análise dos valores experimentados podem ser encontrados no capítulo 6.

(iii). Parâmetro **sched_latency_ns.** Observamos que o melhor resultado obtido foi com o valor 100000000. Este parâmetro representa a latência da preempção para tarefas *CPU-bound*. Mais informações sobre este parâmetro de configuração e análise dos valores experimentados podem ser encontrados no capítulo 6.

(iv). Parâmetro **sched_min_granularity_ns.** Neste parâmetro, observamos que o valor 100000 conduziu a melhores resultados. Como observado no capítulo 6, experimentos foram realizados com os valores 50000, 100000 e 250000. Estes valores foram considerados nos experimentos a partir do referencial teórico (DOMINGO; BAILEY, 2016) (CILIENDO; KUNIMASA, 2008) (PARZIALE et al., 2011) e dos trabalhos relacionados (KRISHNAMURTHY; ROUSKAS, 2015). Mais informações sobre este parâmetro de configuração e análise dos valores experimentados podem ser encontrados no capítulo 6.

(v). Parâmetro **sched_wakeup_granularity_ns.** Com este parâmetro de configuração é possível eliminar a preempção. Conseguimos melhores resultados configurando este parâmetro como 0 (zero) ou 25000. O valor 0 foi sugerido nos trabalhos relacionados (KRISHNAMURTHY; ROUSKAS, 2015). Para fins de experimentos utilizamos o valor 25000. Mais informações sobre este parâmetro de configuração e análise dos valores experimentados podem ser encontrados no capítulo 6.

(vi). Parâmetro **sched_nr_migrate.** Para este parâmetro de configuração, o qual controla quantas tarefas podem ser movidas entre processadores, consideramos os valores 64 ou 128. Nos experimentos, implementamos a carga de trabalho sobre os valores 64, 128, 256, 512 e 1024. Mais informações sobre este parâmetro de configuração e análise dos valores experimentados podem ser encontrados no capítulo 6.

(vii). Utilização de grupos de controle para os processos e *threads* do Java. Mais informações sobre grupos de controles (cpugroups) podem ser encontrados no capítulo 6.

## 9.2 Análise e Resultados Obtidos - Subsistema de Memória Virtual

Os experimentos realizados quanto ao subsistema de gerenciamento de memória virtual, não consideramos como fim último obter redução no tempo de execução da aplicação, mas conceder a vazão necessária quanto à reclamação de memória dos processos do Java e minimizar *overheads* pertinentes.

Os parâmetros de configurações aqui descritos se relacionam com o comportamento do kernel do Linux quanto à reclamação de memória das tarefas, tratamento dos *buffers* de I/O de disco em memória e a granularidade da paginação. Neste contexto, os objetivos primários para os experimentos realizados foram:

- Consideramos garantir a reclamação de memória dos processos e *threads* do Java sem validação de heurística;
- Prover equilíbrio entre as execuções de *threads* do kernel e operações de *buffers* de I/O de disco, minimizando o acesso ao disco;
- Minimizar o *overhead* no tratamento de páginas de memória.

A tabela 19 apresenta os principais os valores dos parâmetros de configurações do Modelo Comum, ou seja, configuração padrão do sistema operacional Linux. Mais informações sobre estes parâmetros de configurações e a análise dos valores experimentados podem ser encontrados no capítulo 7. Na tabela 19, condensamos os valores para o Modelo Comum (MC) e Modelo Ajustado (MA).

Tabela 19 - Configuração de atributos de gerenciamento de memória virtual no Modelo Comum (MC) e Modelo Ajustado (MA).

| **Parâmetro** | **Valor MC** | **Valor MA** | **Comentário** |
|---|---|---|---|
| vm.overcommit_memory | 0 | 1 | Faz com que o kernel atenda a reclamação de memória por meio de heurística, verificando se há memória disponível (0). Adotamos o valor 1, retirando a heurística e sempre atendendo a reclamação de memória |
| vm.swappiness | 60 | 10 | Este parâmetro trata a propensão a swap. Quanto maior o valor, maior a propensão a swap. Minimizamos a propensão a *swap* de forma que *swap* ocorra somente quando 90% da memória RAM for atingida |
| vm.dirty_background_ratio | 10 | 50 | Este parâmetro é um *trade-off*, pois ao mesmo tempo que o número de páginas sujas diminuem, operações de o I/O de disco aumentam. Por meio das referências optamos por adiar operações de I/O. |
| vm.dirty_ratio | 20 | 15 | Controla o descarregamento do *writeback* quando este valor é excedido. Por meio das referências optamos por minimizar *writeback* |
| oom_adj | 0 | -17 | Crivo do OOM_KILLER. Com -17 o processo não poderá ser terminado pelo OOM_KILLER. |
| vm.dirty_writeback_centisecs | 500 | 360000 | Controla a execução da *thread* pdflush para *writeback*. Por meio das referências optamos por adiar a execução da *thread* pdflush. O Ext4 não se beneficia dessa *thread*. |

### 9.2.1 ***Análise dos Resultados no Modelo Ajustado***

Neste conjunto de experimentos utilizamos a execução do código do WordCount para obter dados de análise. Em nossos experimentos utilizamos *scripts* em *shell script* para analisar arquivos do ProcFS (/proc) periodicamente.

Obtivemos dados dos arquivos /proc/meminfo, /proc/slabinfo e /proc/memstat. Observamos principalmente as páginas limpas (ativas) e sujas do sistema (/proc/meminfo) e também a relação da quantidade de páginas sujas e *writeback* uma vez que modificamos o comportamento do kernel para o Modelo Ajustado.

Primeiramente, modificamos o comportamento do kernel quanto ao tratamento de reclamação de memória. No Modelo Comum, quando ocorria a reclamação de memória, havia um processo de heurística para determinar se havia memória RAM disponível. No Modelo Ajustado este comportamento foi alterado de forma que o kernel considera que sempre há memória disponível.

Evitamos a propensão a *swap* pelo fato de que o tempo de acesso de I/O de disco é da ordem milissegundos e o acesso à memória RAM são de nanossegundos. Configuramos a *thread* de kernel em *background*, pdflush, para executar quando 50% da memória RAM for utilizada, procrastinando a execução da *thread* pdflush. No Modelo Comum este valor correspondia a 10% da memória RAM.

Em nossas análises, observamos que o sistema de arquivos Ext4 não se beneficia da *thread* pdflush (desde o kernel 2.6.32) e portanto, o parâmetro vm.dirty_writeback_centisecs foi ajustado para 360000, fazendo com que a *thread* pdflush seja execute somente a cada hora.

Também, para fins de automatização, desenvolvemos um código auxiliar para ajustar o parâmetro oom_adj dos processos do Java para -17 (este valor é valido para todas as *threads* oriundas do processo), fazendo com que os processos do Java não sejam interrompidos pelo crivo do OOM_KILLER.

Nestes ajustes iniciais garantimos que os processos e *threads* do Java obtivessem maior vazão quanto à reclamação de memória e menor *overhead* no tratamento de páginas de memória.

### 9.2.2 *Considerações Pertinentes no Gerenciamento de Memória Virtual*

Foram identificados 37 (trinta e sete) parâmetros de configurações pertinentes ao subsistema de memória virtual. A partir da análise do referencial teórico e documentações oficiais, catalogamos como relevância de investigação 16 (dezesseis) parâmetros.

Desse conjunto de 16 (dezesseis) parâmetros, identificamos que quatro em particular e um determinado atributo do gerenciamento da tarefa, o OOM (*Out Of Memory*), foram considerados relevantes na perspectiva deste trabalho.

As páginas grandes (*Huge Pages*) não foram contempladas neste trabalho. Isto envolveria análises com *collect garbage* da JVM que extrapoloria o escopo proposto.

A tabela 20 apresenta uma visão geral dos parâmetros pertinentes ao subsistema de memória virtual. Na coluna **Estado**, tem-se **A**, para os parâmetros que foram **Analisados** por meio do referencial teórico, documentação oficial e outras fontes, **T**, para os parâmetros que foram **Testados** por meio dos experimentos realizados e **U**, para os parâmetros que foram **Utilizados** por impactar no desempenho do sistema e compor o Modelo Ajustado. Há que se observar que **todos** os parâmetros foram analisados, mas nem todos foram testados nos experimentos e/ou utilizados.

Os parâmetros **Testados**, mas que não foram **Utilizados** representaram pouca ou nenhuma melhoria no desempenho ou relação direta com o objeto de estudo deste trabalho. Mais informações sobre os principais parâmetros de configurações que consideramos relevantes no impacto do desempenho do sistema e análise dos valores experimentados podem ser encontrados no capítulo 7. No anexo estão descritos todos os parâmetros que foram analisados e não estão descritos na tabela 20.

Tabela 20 - Quadro geral dos parâmetros de configurações quanto ao subsistema de memória virtual.

| **Parâmetro** | **Estado** | **Descrição** |
|---|---|---|
| vm.dirty_background_bytes | A/T | Contém a quantidade de memória arcada como suja (dirty) na qual o thread de kernel iniciará o processo de writeback. |
| vm.dirty_background_ratio | A/T/U | Representa a porcentagem total da quantidade de memória livre e recuperável. Quando a quantidade de *Page Cache* dirty excede essa porcentagem, as *threads* de writeback iniciam a gravação de memória dirty. |
| vm.dirty_bytes | A/T | Contém a quantidade de memória suja na qual um processo que gera gravações em disco inicia o writeback.O valor mínimo permitido para dirty_bytes são duas páginas (em bytes). Cada página possui 4096 bytes. Valores inferiores a este limite será ignorado. |
| vm.dirty_expire_centisecs | A/T | Define quando os dados sujos são velhos o suficiente para serem elegíveis para escrita pelas *threads* de flush do kernel. |
| vm.dirty_ratio | A/T/U | A quantidade máxima de memória que pode ser preenchida com páginas sujas antes de ser escritas no disco. |
| vm.dirty_writeback_centisecs | A | Define quando as *threads* de flush do kernel irão periodicamente acordar e gravar dados "velhos"no disco. |
| vm.drop_caches | A | Manipula a liberação de memória cache, sendo 1, para liberar *Page Cache*, 2, para liberar objetos de cache reclamados (incluindo dentries e inodes) e 3, para liberar tanto Page Cache quanto objetos de cache reclamados. |
| vm.min_free_kbytes | A | Controla a quantidade de memória que é mantida livre para uso extraordinário, incluindo alocações "atômicas"(aquelas que não podem esperar para recuperar). O valor padrão depende da quantidade de RAM. |
| vm.nr_hugepages | A | Altera o tamanho mínimo do pool de *hugepage*. |
| vm.nr_overcommit_hugepages | A | Altera o tamanho máximo do pool de hugepage. O máximo é definido como nr_hugepages + nr_overcommit_hugepages. |
| vm.overcommit_kbytes | A | Se a variável overcommit_memory for definido como 2, o espaço de endereço alocado não pode exceder o espaço de swap mais essa quantidade de RAM física. |
| vm.overcommit_memory | A/T/U | Define como o kernel atenderá a reclamação de memória. |
| vm.overcommit_ratio | A | Representa o valor a ser considerado quando o inteiro 2 (dois) é definido na variável vm.overcommit_memory. |
| vm.page-cluster | A | Controla o número de páginas consecutivas que podem ser lidas na memória de swap em uma única tentativa. |
| vm.swappiness | A/T/U | Esse controle é usado para definir quão agressivamente o kernel troca a memória anônima relativa a *Page Cache* e outros caches. Aumentar este valor aumenta a quantidade de swap. |
| vm.vfs_cache_pressure | A/T/U | Esta variável controla a tendência do kernel em recuperar a memória que é usada para o cache do VFS (Virtual File System) em contrapartida ao *Page Cache* e swap. |

## 9.3 Análises e Resultados Obtidos - Subsistema de Arquivos e I/O de Disco

Em um dos trabalhos relacionados (REZGUI et al., 2014), foi observado que todos os agendadores de I/O de disco do Linux possuem desempenho muito próximo. Esta obervação poderia permitir concluir, que de certa forma, a escolha do agendador não influencia no desempenho.

Sendo assim, buscamos inicialmente realizar uma pesquisa para investigar se outros trabalhos chegaram ao mesmo resultado e em caso positivo, não iríamos realizar experimentos no subsistema de de I/O de disco. Com os estudos e análises do referencial teórico (PRATT; HEGER, 2004) (ALVES, 2009) encontramos trabalhos com resultados diferentes que justificavam experimentos no subsistema de I/O de disco neste trabalho.

Sendo assim, nos trabalhos relacionados, descrevemos parte das análises de outros trabalhos (PRATT; HEGER, 2004) (ALVES, 2009) (DOMINGO; BAILEY, 2016) corroborando a relevância de investigaçã e pesquisa e também no capítulo 8, dissertamos sobre parâmetros de configurações do subsistema de I/O de disco que influenciam no desempenho.

Também, analisamos a relação do sistema de arquivos quanto ao sistema de armazenamento e o subsistema de I/O de disco. Exclusivamente quanto ao sistema de arquivos, por meio das análises dispostas (ALVES, 2009), encontramos resultados quanto ao acesso a uma grande quantidade de arquivos ((ALVES, 2009) não especifica a quantidade de arquivos) e que os experimentos realizados favoreceram o desempenho.

O sistema de arquivos com a opção de atime ativada obteve um tempo de execução de 406 segundos, enquanto que o mesmo sistema de arquivos com a opção de atime desativada obteve-se o tempo de execução de 390 segundos, um fator de 4% de desempenho (ALVES, 2009).

Em outra análise (ALVES, 2009), quanto às políticas de *journaling*, a criação de um arquivo de 1.6 GB culminou em um tempo de execução de 21,9 segundos com a política *Writeback*, 23,14 segundos com a política *Ordered* e 55,41 segundos com a política *Journal*. Neste mesmo procedimento, analisando a taxa de transferência, obteve-se um *throughput* de 74,47 MB/s com a política *Writeback*, 68,96 MB/s com a política *Ordered* e 28,5 MB/s com a política *Journal*.

Portanto, justificamos aqui a relevância da pesquisa realizada por meio do referencial teórico e dos experimentos realizados neste trabalho.

### 9.3.1 *Análise dos Experimentos no Modelo Comum e Modelo Ajustado*

Como descrito na metodologia, utilizamos o TestDFSIO e o TIOBench para carga de trabalho com *threads* no ambiente de experimentos. A tabela 21 apresenta os dados gerais da partição /hadoop.

Tabela 21 - Configuração da Partição /hadoop no Modelo Comum/Modelo Ajustado.

| Partição | Montagem | File System | Tempo de Acesso | Journal |
|---|---|---|---|---|
| /dev/sda1 (Comum) | /hadoop | ext4 | relatime | ordered |
| /dev/sda1 (Ajustado) | /hadoop | ext4 | noexec,noatime, nodiratime,nobarrier | writeback |

Na tabela 22 consolidamos os principais parâmetros de configurações do Modelo Ajustado e Modelo Comum. Mais informações quanto aos parâmetros analisados e considerados relevantes para este trabalho no contexto de desempenho no subsistema de I/O de disco podem ser encontradas no capítulo 8. Informações de análises que consideramos relevantes de outros trabalhos podem ser encontradas descritas no referencial teórico e nos trabalhos relacionados.

Tabela 22 - Comparação dos parâmetros de configurações do Modelo Comum e do Modelo Ajustado.

| Modelo Comum | | Modelo Ajustado | |
|---|---|---|---|
| **Item** | **Valor** | **Item** | **Valor** |
| I/O Secheduler | Deadline | I/O Secheduler | NOOP, Deadline, CFQ |
| nr_requests | 128 | nr_requests | 64/256/512/1024 |
| read_ahead_kb | 128 | read_ahead_kb | 256 |
| rq_affinity | 1 | rq_affinity | 1 |

### 9.3.2 *Experimentos de Operações de I/O Envolvendo Leitura/Escrita*

Nesta seção estaremos analisando os experimentos realizados em uma visão comparativa. Nestes experimentos realizamos um conjunto de cinco execuções, porém, utilizamos aqui os dados referentes a uma única execução. Também, não realizamos o reinício da máquina virtual durante as execuções. Nestes experimentos utilizamos um *script* para automatizar a remoção das estruturas de cache e arquivos gerados.

Nos experimentos do subsistema de arquivos e I/O de disco realizamos a execução do TIOBench na partição /hadoop. As execuções foram realizadas por meio de um *script* que realiza os seguintes passos (o mesmo processo foi utilizado para o TestDFSIO):

(i) Remove as estruturas de cache de I/O;

(ii) Seleciona o agendador de I/O (Deadline, NOOP e CFQ);

(iii) Executa o TIOBench;

(iv) Remove os arquivos gerados.

A tabela 23 apresenta os resultados obtidos com a execução do TIOBench analisando o comportamento do *throughput* de I/O de disco ajustando o parâmetro de configuração nr_requests. Nestes experimentos realizamos operações de I/O sobre 1000 (mil) arquivos de 10 MB utilizando 256/300 *threads*.

Tabela 23 - Experimento de Leitura/Escrita - TIOBench

| **Execução TIOBench - Modelo Comum (Throughput)** | | | | |
|---|---|---|---|---|
| **I/O Sched.** | **Write** | **Random Write** | **Read** | **Random Read** |
| NOOP | 385.504 MB/s | 29.910 MB/s | 104.827 MB/s | 12.811 MB/s |
| Deadline | 236.835 MB/s | 26.524 MB/s | 106.137 MB/s | 12.754 MB/s |
| CFQ | 228.268 MB/s | 13.582 MB/s | 55.667 MB/s | 17.369 MB/s |

Analisando os dados da tabela 23, observarmos que o agendador de I/O de disco NOOP obteve melhor desempenho para escrita. Nestes experimentos o valor do parâmetro de configuração nr_requests foi mantido como padrão (128). Observamos também que o agendador de I/O de disco Deadline obteve melhor desempenho na leitura sequencial e o agendador de I/O de disco CFQ apresentou resultados superiores somente para leitura randômica. As figuras 9, 10 e 11 apresentam esses resultados por meio de gráficos.

Figura 9 - Resultados obtidos com o agendador de I/O de disco NOOP com o parâmetro nr_requests definido como 128 (padrão do sistema).

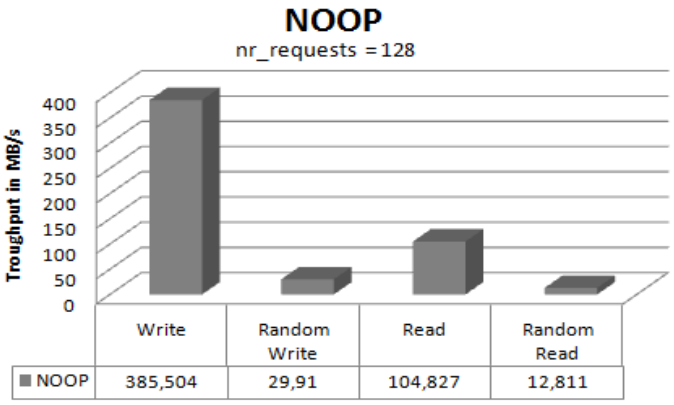



Figura 10 - Resultados obtidos com o agendador de I/O de disco Deadline com o parâmetro nr_requests definido como 128 (padrão do sistema).

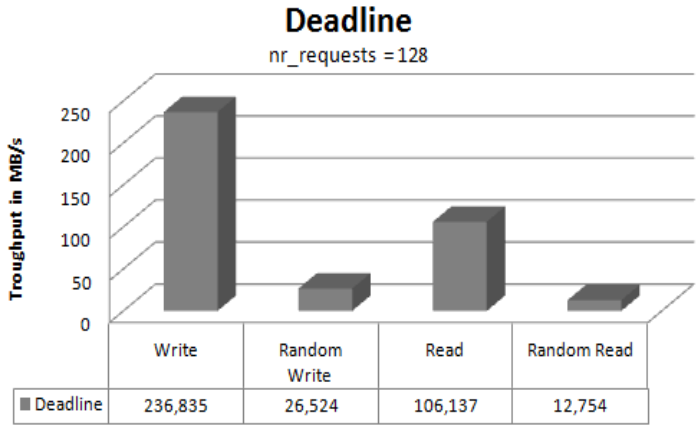
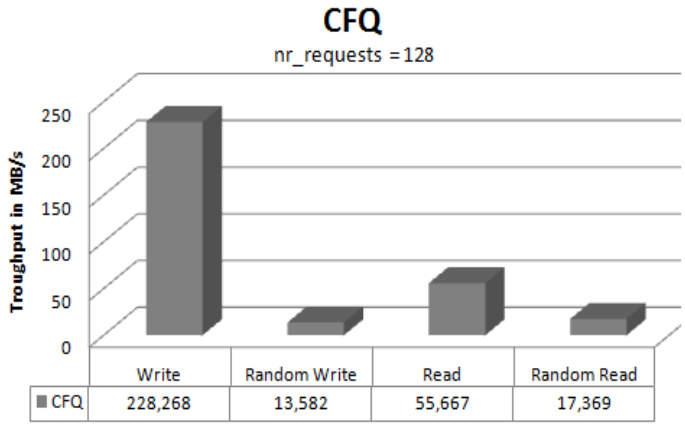


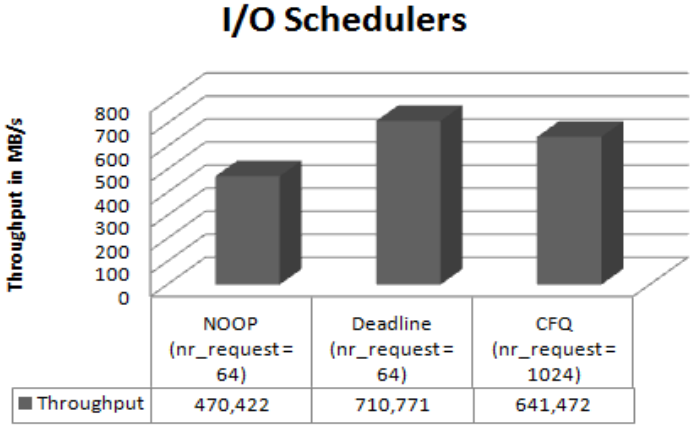
Figura 11 - Resultados obtidos com o agendador de I/O de disco CFQ com o parâmetro nr_requests definido como 128 (padrão do sistema).




A figura 12 apresenta um gráfico comparativo com os três agendadores de I/O de disco com ajustes realizados no parâmetro de configuração nr_requests. Para estes experimentos realizamos um conjunto de cinco experimentos para cada valor ajustado do parâmetro de configuração nr_requests. Estes valores foram 32, 64, 128, 256, 512, 1024, 2048 e 4096. O capítulo 8 apresenta mais informações sobre o parâmetro nr_requests e a motivação da escolha de tais valores.

O gráfico da figura 12 apresenta exclusivamente os melhores resultados obtidos para o valor ajustado no parâmetro de configuração nr_requests.

Figura 12 - Quadro comparativo dos melhores resultados dos agendadores de I/O com o parâmetro nr_requests com valores de entrada de 32, 64, 256, 512, 1024, 2048 e 4096.



Como pode ser observado, o melhor *throughput* baseado nos ajustes realizados no parâmetro de configuração nr_requests, temos o valor 64 com o agendador de I/O de disco Deadline, o valor 64 com o agendador agendador de I/O de disco NOOP e 1024 com o agendador de I/O de disco CFQ. Em nossos experimentos observamos que valores acima de 1024 produziram os piores resultados em todos os agendadores de I/O de disco. Também observamos que a partir do conjunto de valores analisados, em todos os casos observamos que o padrão do sistema (128) **não** obteve resultados melhores que os valores ajustados.

Como parte de nossos experimentos, consideramos a realização da carga de trabalho **exclusivamente** com os ajustes pertinentes ao sistema de arquivos Ext4 (estes ajustes foram realizados diretamente na partição /hadoop). As figuras 13 e 14 apresentam os resultados obtidos. Nestes experimentos utilizamos o TestDFSIO e TIOBench realizando operações de I/O sobre 1000 (mil) arquivos de 10 MB utilizando 256/300 *threads*.

Figura 13 - Quadro comparativo dos resultados obtidos no Modelo Ajustado comparado com o Modelo Comum utilizando nos experimentos o TestDFSIO.

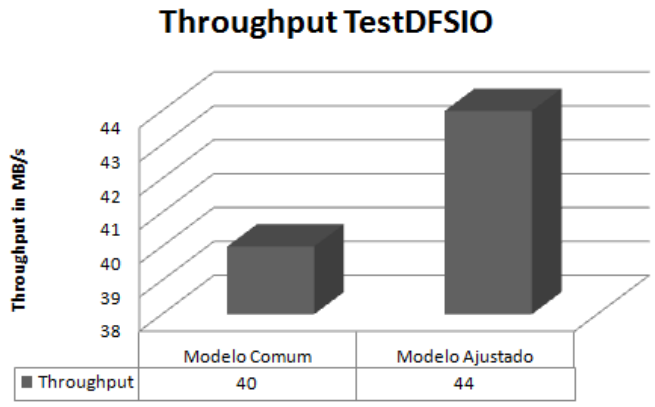
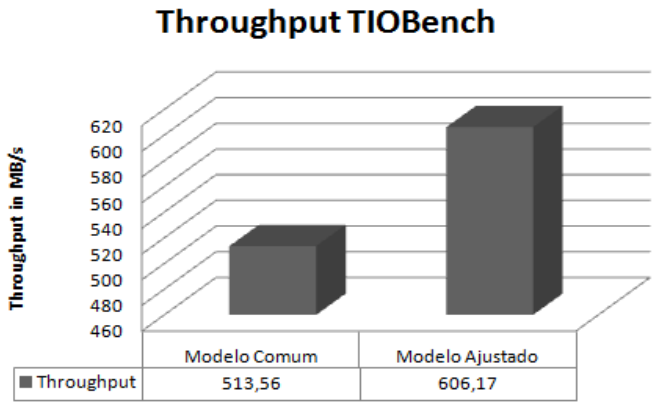


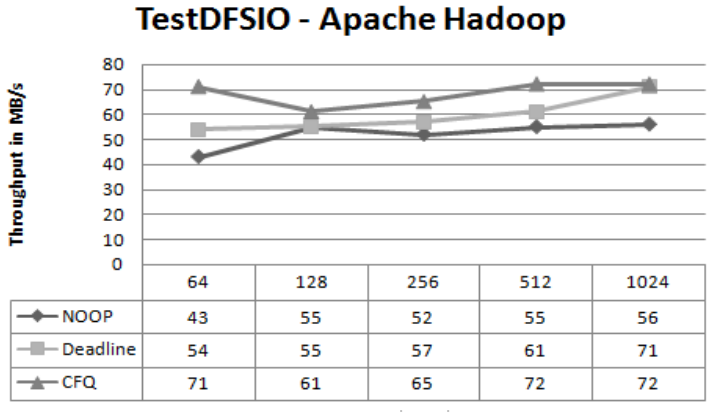
Figura 14 - Quadro comparativo dos resultados obtidos no Modelo Ajustado comparado com o Modelo Comum com experimentos utilizando o TIOBench.




Obeservamos que somente com as modificações nos parâmetros de configurações do Ext4 na partição /hadoop, com tudo o mais constante, conseguimos uma melhoria com fator de 10% no *throughput* com experimentos utilizando o TestDFSIO e uma melhoria com fator de 18% com experimentos utilizando o TIOBench.

A figura 15 apresenta os resultados agrupando-os por agendador de I/O de disco. Nestes resultados estão contemplados ajustes realizados nos parâmetros de configurações do sistema de arquivos Ext4 na partição /hadoop.

Figura 15 - Quadro comparativo dos resultados obtidos no Modelo Ajustado comparado os agendadores de I/O utilizando o TestDFSIO.



Observamos na figura 15 que o agendador de I/O de disco CFQ obteve melhor resultado em todos os valores do parâmetro de configuração nr_requests utilizados. Os agendadores de I/O de disco NOOP e Deadline obtiveram melhores resultados com o valor do parâmetro de configuração nr_requests ajustado para 1024.

Também, observamos nos experimentos realizados quantos aos agendadores de I/O de disco é os melhores resultados (dependendo do agendador) pode ser conseguido com o parâmetro de configuração nr_requests para os valores 64, 256, 512 e 1024. Em nossos experimentos, valores a partir de 2048 conduziram a piores resultados, apesar de não serem apresentados aqui, pois o objetivo do trabalho é encontrar os melhores resultados. Por fim, o valor padrão (128), no Modelo Comum, para o parâmetro de configuração nr_requests conduziu a resultados com desempenho inferior aos ajustes realizados.

Também observamos que em sistemas que possuem um subsistema de RAID, o valor de read_ahead_kb pode ser ajustado multiplicando o número de discos pelo valor padrão (128), o que pode otimizar as leituras em um *buffer* baseado em *prefetch*. No caso de virtualização, resultados satisfatórios foram alcançados com o dobro do valor padrão (portanto, 256), porém, não muito significativo.

As concretizações da pesquisa realizada a partir do referencial teórico e experimentos realizados no contexto do subsistema de arquivos e I/O de disco, podem ser traduzidas da seguinte forma:

- Se a aplicação for utilizada para realizar leituras mais constantes que escritas, o agendador de I/O Deadline pode ser a opção mais viável.
- Se o sistema de armazenamento subjacente possui mecanismos intrínsecos de tratamento de blocos de I/O, como RAID e redes de armazenamento remoto, o agendador NOOP pode ser a opção mais viável.
- Se há muitas aplicações concorrentes e torna-se necessário uma fatia justa no tratamento de I/O, o agendador CFQ é a opção mais viável.
- Se for necessário um tratamento mais granular na prioridade de I/O das tarefas, o agendador CFQ é a opção mais viável.
- Em todos os casos, as configurações descritas neste trabalho sobre as opções de montagem da partição para a aplicação crítica no sistema de arquivos Ext4, são imprescindivelmente necessárias.

Por fim e não menos importante, também foi observado o aspecto da tecnologia do sistema de arquivos, ou seja, o conjunto de mecanismos que estes oferecem para o embarco das aplicações. Neste contexto, fica claro que os sistemas de arquivos devem evoluir no que tange ao tratamento massivo de dados. O sistema de arquivos atual, o Ext4, atende as aplicações de âmbito geral quanto ao tamanho de bloco, escalabilidade mínima e desempenho.

Por outro lado, como observado em (KERNEL.ORG, 2017) e (HAT, 2017), o tamanho máximo de arquivos manipulados pelo Ext4 varia dado o tamanho do bloco, o que implica que dependendo das características da aplicação o tamanho do bloco utilizado pode reduzir a capacidade máxima do sistema em lidar com o armazenamento.

Um dos sistemas de arquivos promissores na era de *Big Data* e no âmbito do Apache Hadoop e aplicações de missão crítica é o Oracle Solaris ZFS (*Zettabyte File System*), o qual alinha com as características do *Big Data*. O Oracle Solaris ZFS possui suporte a tecnologias como (KIRKLAND, 2016):

- Confiabilidade de dados com *Self-Healing*.
- Reconstrução *on-line* do array de discos.
- Desduplicação e compactação.
- *Striping* dinâmico para maximização de *throughput*.
- COW (*Copy On Write*), maximizando as gravações sequenciais.
- Tamanhos de blocos ajustáveis dinamicamente à carga de trabalho.

Também, (BONWICK, 2008), ainda cita-se outras características particulares do ZFS para provisionamento de desempenho, flexibilidade e gerenciamento:

- Sistema nativo de LVM.
- Storage Pools.
- Sistema de *snapshot* nativo.
- *Pipelined* I/O.
- Mecanismo de *prefetch* inteligente.
- Alocação dinâmica de inodes.
- Formas granulares de gestão.

Inicialmente, devido a questões de conflito de licença no Linux, o Oracle Solaris ZFS era suportado como FUSE (*File System Userspace*) e também como um *port* nativo de integração com o kernel do Linux (JONES, 2011). Nas versões atuais do Ubuntu Server, o ZFS é plenamente suportado, exceto como sistema de arquivos raiz, e a arquitetura deve ser de 64-Bits (KING, 2016).

Neste trabalho não foi abordado experimentos com o sistema de arquivos ZFS pelo fato de este não ser parte padrão da base operacional Linux e o cerne do trabalho não implica em comparações entre sistema de arquivos.

## 9.4 **Análises Conclusivas e Resultados**

A tabela 24 apresenta os parâmetros de configurações analisados no subsistema de tarefas e os valores correspondentes em que obtivemos melhores resultados.

Tabela 24 - Principais parâmetros de configurações do subsistema de tarefas que mais influenciaram no desempenho.

| **Parâmetro de Configuração** | **Ajuste** | **Descrição** |
|---|---|---|
| sched_migration_cost_ns | 50000000 | Quantidade de tempo após a última execução em que uma tarefa é considerada *"cache hot"* nas decisões de migração. |
| sched_autogroup_enabled | 1 | Habilita a utilização de grupos de CPU para gestão de recursos. |
| sched_nr_migrate | 64 | Quantas tarefas podem ser movidas entre processadores através de interrupções de software (softirq). |
| sched_wakeup_granularity_ns | 0/25000 | A granularidade de preempção de "acordar". |
| sched_min_granularity_ns | 100000 | Granularidade de preempção mínima para tarefas *CPU-bound*. |
| sched_latency_ns | 1000000 | Latência da preempção direcionada para tarefas *CPU-bound*. |
| fs.file-max | 665536 | Número máximo de arquivos manipulados por todo o sistema. |
| nofile | 665536 | Número máximo de arquivos que podem abertos no sistema. |

Também, consideramos como relevante as seguintes questões:

(i). Realizar a afinidade de processador, para minimizar o *overhead* da troca de contexto;

(ii). Utilização de grupos de controle de CPU;

(iii). Desabilitar serviços desencessários no sistema.

Privilegiando o **desempenho**, ainda sugerimos o seguinte:

(i). Destivar recursos de segurança como SELinux e AppArmor. Esta decisão deve ser realizada com parcimônia. O SELinux é fundamentalmente importa para a segurança do Linux;

(ii). Desativar *TCP Offload Engine* (TOE) da interface de rede. Em ambiente de virtualização esta é uma prática que deve ser adotada quando requer-se trafégo intenso de rede. Neste trabalho foi contemplado o Apache Hadoop como *Single Node*, ou seja, não contemplamos um ambiente de cluster. Também, o subsistema de I/O de rede não foi analisado neste trabalho;

(iii). Desativar mecanismos de auditoria (*audit*). Apesar de ser um recurso fundamental para rastreamento do kernel no contexto de segurança, escritas constantes são realizadas.

A tabela 25 apresenta os principais parâmetros de todo o conjunto analisado em que as modificações destes implicavam no comportamento do desempenho quanto ao subsistema memória virtual.

Tabela 25 - Principais parâmetros de configurações do subsistema de memória virtual que mais influenciaram no desempenho.

| **Parâmetro de Configuração** | **Ajuste** | **Descrição** |
|---|---|---|
| vm.overcommit_memory | 1 | Faz com que o kernel atenda a reclamação de memória por meio de heurística, verificando se há memória disponível. |
| vm.swappiness | 10 | Este valor trata a propensão a swap. Quanto maior o valor, maior a propensão a swap. |
| vm.dirty_background_ratio | 50 | Esta configuração é um trade-off, ao mesmo tempo que o número de páginas sujas diminuem, o acesso ao disco aumenta. |
| vm.dirty_ratio | 15 | Controla o descarregamento do writeback quando este valor é excedido. |
| oom_adj | -17 | Processos do Apache Hadoop não estão nas mesmas condições que qualquer outro processo no sistema para sofrer interrupção em caso de OOM. |
| vm.dirty_writeback_centisecs | 360000 | Parâmetro que permite procrastinar a *thread* pdflush, mitigando tal *overhead*. |

Um aspecto importante que observamos nos parâmetros de configurações do subsistema de memória virtual é que apenas cinco influenciaram diretamente no desempenho (vm.swappiness, vm.dirty_background_ratio, vm.dirty_ratio e vm.dirty_writeback_centisecs). Um outro que permite garantir que a aplicação sempre obtenha memória para alocação (vm.overcommit_memory). Por fim, um parâmetro que garante que a aplicação Apache Hadoop não seja interrompida no exaurir da memória, sacrificando outras tarefas, exceto as tarefas do Apache Hadoop no caso de OOM_Killer (oom_adj).

A tabela 26 apresenta os principais parâmetros de todo o conjunto analisado em que a modificação destes implicavam no comportamento do desempenho do Apache Hadoop quanto ao subsistema de arquivos e de I/O de disco.

Tabela 26 - Principais parâmetros de configurações do subsistema de arquivos e I/O de disco que mais influenciaram no desempenho.

| **Parâmetro de Configuração** | **Valor** | **Descrição** |
|---|---|---|
| nr_requests | 64/256/512/1024 | Cada fila de solicitações tem um limite no número total de descritores de solicitação que podem ser alocados para cada I/O de leitura e escrita. Por meio das referências realizamos experimentos com os valores 64, 256, 512, 1024, 2048 e 4096. |
| read_ahead_kb | 256 | Executa um algoritmo de leitura antecipada (*read-ahead*), em que mais dados do que o solicitado pela tarefa são lidos a partir do disco. Por meio das referências realizamos experimentos com os valores 256, 4096 e 8192. |
| rq_affinity | 1 | A definição de rq_affinity para 1 faz com que o kernel forneça as respostas das solicitações de I/O à CPU na qual a requisição de I/O foi emitida. |

Em nossos experimentos, de forma geral, quanto ao subsistema de arquivos e I/O de disco, observamos:

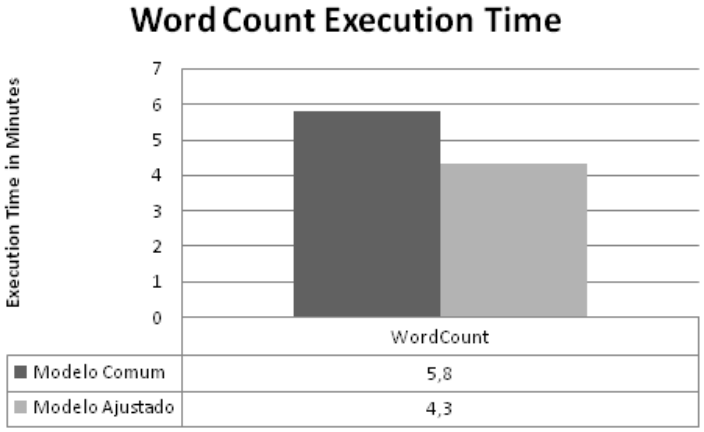
(i). Que nos experimentos com os agendadores de I/O de disco NOOP, Deadline e CFQ, identificamos que o parâmetro nr_requests possui impacto considerável na escolha dos agendadores de I/O de disco.

(ii). Quem nas pesquisas que realizamos, constatamos que o desempenho do agendador de I/O de disco está diretamente relacionado com a característica da carga de trabalho, o tipo de operação de I/O (leitura/escrita) e do subsistema de armazenamento.

(iii). Observamos que modificações nos parâmetros de configurações da partição do Apache Hadoop com o sistema de arquivos Ext4, como noatime, nodiratime, nobarrier e mecanismo de *journaling* com *writeback* são fundamentalmente relevantes para obter melhor desempenho, principalmente para escrita.

A figura 16 apresenta um gráfico comparativo a partir dos resultados obtidos com a execução do WordCount quanto ao tempo de execução. Como observado, em relação aos parâmetros de configurações padrão do sistema (Modelo Comum), obtivemos uma melhoria de 34% no tempo do código do WordCount.

Figura 16 - Resultados obtidos com modificações nos parâmetros de configurações do subsistema de tarefas e memória virtual na execução do WordCount.



No subsitema de memória virtual os parâmetros de configurações garantiram a reclamação de memória para as tarefas do Apache Hadoop e diminuíram o *overhead* da propensão de acesso à memória de troca (*swap*). Também garantimos que as tarefas do Java não passem pelo crivo do OOM_KILLER

Nas figuras 17 e 18 demonstram o ganho de desempenho no *throughput* com os ajustes nos parâmetros de configurações do subsistema de arquivos e de I/O de disco. Com o TestDFSIO obtivemos uma melhoria de 52% e com o TIOBench, obtivemos uma melhoria de 18,5%.

Figura 17 - Resultados obtidos com modificações nos parâmetros de configurações do subsistema de arquivos e I/O de disco.

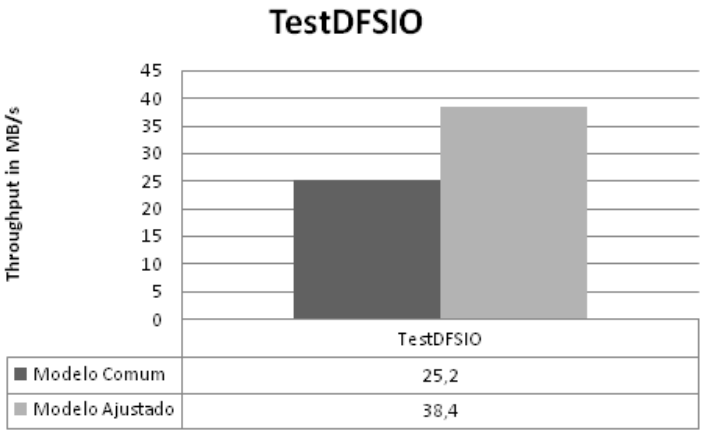
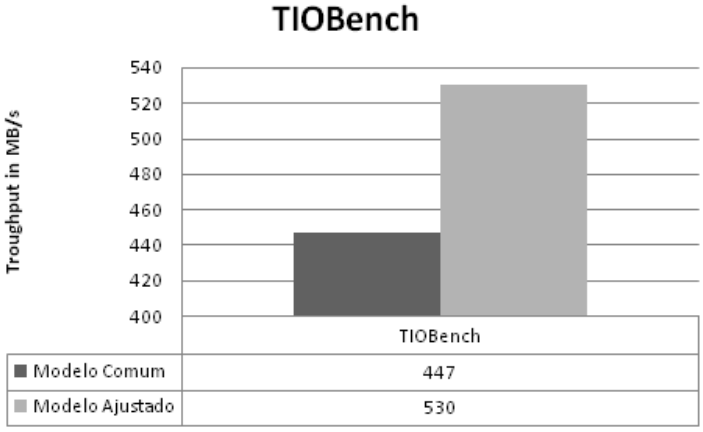



Figura 18 - Resultados obtidos com modificações nos parâmetros de configurações do subsistema de arquivos e I/O de disco.



# 10 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho analisou os principais parâmetros de configurações no sistema operacional Linux quanto aos subsistemas de tarefas, memória virtual, sistema de arquivos e I/O de disco. Foram analisados mais de 600 (seiscentos) parâmetros de configurações dos principais subsistemas do kernel do Linux.

Deste conjunto, foi estudado um subconjunto máximo de parâmetros que influenciam diretamente no desempenho do Apache Hadoop e aplicações de missão crítica com características semelhantes. Este subconjunto foi catalogado e experimentado em ambiente de virtualização.

No contexto dos experimentos, foram ralizados mais de 400 (quatrocentas) cargas de trabalho e execuções pontuais que permitiram observar e concluir que **(i)** os ajustes de parâmetros de configurações influenciam indubitavelmente no desempenho das aplicações de missão crítica e **(ii)** os parâmetros de configurações padrão do sistema não fornecem melhor desempenho e escalabilidade para as aplicações de missão crítica.

Portanto, é indubitável que são necessários ajustes de tais parâmetros de configurações para oferecer o arcabouço possível quanto ao desempenho e escalabilidade das aplicações de missão crítica em ambientes de servidores Linux.

Também, este trabalho permitiu lançar luz sobre como os parâmetros de configurações dos subsistemas do kernel do Linux são estruturados e sua semântica. Isto favorece as empresas e instituições compreenderem como tais parâmetros de fato influenciam no comportamento das aplicações de missão crítica na perspectiva do subsistema em que deseja-se aplicar ajustes de desempenho.

Para trabalhos futuros uma pesquisa estendida ao subsistema de rede permitirá realizar ajustes de parâmetros de configurações para otimizar o tratamento de rede em ambiente de virtualização para aplicações de missão crítica. No subsistema de arquivos e I/O de disco, é pertinente uma análise exclusiva com o sistema de arquivos ZFS em comparação com o Ext4.

# REFERÊNCIAS


ALMEIDA, V. A.; MENASCE, D. A. ***Planejamento De Capacidade Para Serviços Na Web***. [S.l.]: Campus, 2003.

ALVES, M. M. ***Linux Performance e Monitoramento***. 1. ed. [S.l.]: Brasport, 2009. v. 1.

ANTYPAS, V.; ZACHEILAS, N.; KALOGERAKI, V. **Dynamic Reduce Task Adjustment for Hadoop Workloads**. *Panhellenic Conference on Informatics (PCI)*, n. 203-208, p. 6, October 2015.

BERGNER, P. et al. ***Performance Optimization and Tuning Techniques for IBM Power Systems Processors Including IBM POWER8***. 2. ed. [S.l.]: IBM Red Book, 2015. v. 1.

BONWICK, J. ***ZFS - The Last Word in File Systems***. 2008. Internet. Disponível em: ⟨https://www.snia.org/sites/default/orig/sdc_archives/2008_presentations/monday\/JeffBonwick-BillMoore_ZFS.pdf⟩.

BORTHAKUR, D. ***The Hadoop Distributed File System: Architecture and Design***. http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html, 2016.

BOVET, D. P. ***Understanding the Linux Kernel, Third Edition***. 3. ed. [S.l.]: O'Reilly Media, 2005.

CHO, J.-Y. et al. **On the Core Affinity and File Upload Performance of Hadoop**. *International Symposium on DIStributed Computing (DISC)*, n. 25-30, p. 6, November 2013.

CILIENDO, E.; KUNIMASA, T. ***Linux Performance and Tuning Guidelines***. 2. ed. [S.l.]: IBM Red Books, 2008. v. 1.

DOMINGO, D.; BAILEY, L. ***Red Hat Enterprise Linux 6 Performance Tuning Guide: Optimizing subsystem throughput in Red Hat Enterprise Linux 6 Edition 4.0***. 1. ed. [S.l.]: Red Hat, Inc. and others., 2016. v. 1.

FUJISHIMA, E.; YAMAGUCHI, S. **Dynamic File Placing Control for Improving the I/O Performance in the Reduce Phase of Hadoop**. *International Conference on Ubiquitous Information Management and Communication (IMCOM)*, n. 1-7, p. 7, January 2016.

GORMAN, M. ***Understanding The Linux Virtual Memory Manager***. [S.l.]: Bruce Peren's Open Book Series, 2009.

HAT, R. ***Red Hat Enterprise Linux technology capabilities and limits***. 2017. Disponível em: ⟨https://access.redhat.com/articles/rhel-limits⟩.

JONES, M. T. ***Run ZFS on Linux***. 2011. Internet. Disponível em: ⟨https://www.ibm.com/developerworks/library/l-zfs/⟩.

KERNEL.ORG. ***Ext4 Disk Layout***. 2017. Internet. Disponível em: ⟨https://ext4.wiki.kernel.org/index.php/Ext4_Disk_Layout⟩.

KING, C. I. ***ZFS***. 2016. Internet. Disponível em: ⟨https://wiki.ubuntu.com/Kernel/Reference/ZFS⟩.

KIRKLAND, D. ***ZFS***. 2016. Internet. Disponível em: ⟨https://wiki.ubuntu.com/ZFS⟩.

KRISHNAMURTHY, R.; ROUSKAS, G. N. **On the Impact of Scheduler Settings on the Performance of Multi-Threaded SIP Servers**. *Communications QoS, Reliability and Modeling Symposium*, n. 6175-6180, p. 6, 2015.

LI, C. et al. **AACT - An Adaptive Auto-Configuration Tool for Hadoop**. *International Conference on Engineering of Complex Computer Systems (ICECCS)*, n. 69-72, p. 4, 2014.

LOUGHRAN, S. ***Powered by Apache Hadoop***. 2016. Apache Software Foundation. Disponível em: ⟨https://wiki.apache.org/hadoop/PoweredBy⟩.

LOVE, R. ***Linux Kernel Development***. 1. ed. [S.l.]: Addison-Wesley Professional, 2010. v. 1.

NAIR, S.; MEHTA, J. **Clustering with Apache Hadoop**. *International Conference and Workshop on Emerging Trends in Technology (ICWET)*, p. 505–509, February 2011.

NEGUS, C.; BRESNAHAN, C. ***Linux Bible 8th Edition***. 9. ed. [S.l.]: John Wiley & Sons, 2015. v. 1.

NEMETH, E.; SNYDER, G.; HEIN, T. R. ***Manual Completo do Linux: Guia do Administrador de Sistemas***. 4. ed. [S.l.]: Pearson Prentice Hall, 2010. v. 1.

PARZIALE, L. et al. ***Linux on IBM System z Performance Measurement and Tuning***. 1. ed. [S.l.]: IBM Red Books, 2011. v. 1.

PATTERSON, D. A.; HENNESSY, J. L. ***Arquitetura de computadores: uma abordagem quantitativa***. 5. ed. [S.l.]: Editora Campus, 2014. v. 1.

PRATT, S. L.; HEGER, D. A. ***Workload Dependent Performance Evaluation of the Linux 2.6 I/O Schedulers***. 2004. Proceedings Of The Linux Symposium 2004 Volume TWo. Disponível em: ⟨https://www.kernel.org/doc/ols/2004/ols2004v2-pages-139-162.pdf⟩.

PRODANOV, C. C.; FREITAS, E. C. de. ***Metodologia do Trabalho Científico:*** *Métodos e Técnicas da Pesquisa Trabalho Acadêmico*. 2. ed. [S.l.]: Editora Feevale, 2013.

RAMOS, A. ***Infraestrutura Big Data com Opensource***. 1. ed. [S.l.]: Ciência Moderna, 2015. v. 1.

REZGUI, A. et al. **Evaluation of Linux I/O Schedulers for Big Data Workloads**. *IEEE Fourth International Conference on Big Data and Cloud Computing*, n. 227-234, p. 8, 2014.

RIEL, R. van; MORREALE, P. W. ***Documentation for /proc/sys/vm/***. 2008. Kernel.Org. Disponível em: ⟨https://www.kernel.org/doc/Documentation/sysctl/vm.txt⟩.

SCHWARTZ, B.; ZAITSEV, P.; TKACHENKO, V. ***High Performance MySQL***. 3. ed. [S.l.]: OReilly Media, 2012.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. ***Fundamentos de Sistemas Operacionais - Princípios Básicos***. 1. ed. [S.l.]: Editora LTC, 2015. v. 1.

SOMASUNDARAM, G.; SHRIVASTAVA, A. ***Armazenamento e gerenciamento das informações: como armazenar, gerenciar e proteger informações digitais.*** [S.l.]: Bookman, 2011.

SOUZA, D. A. A. de. ***FreeBSD - O poder dos servidores em suas mãos***. 1. ed. [S.l.]: Novatec Editora, 2009.

STALLINGS, W. ***Arquitetura e organização de computadores***. 8. ed. [S.l.]: Pearson, 2010.

STUART, B. L. ***Princípios de sistemas operacionais: projetos e aplicações***. [S.l.]: Cengage Learning, 2010.

TANENBAUM, A. S. ***Sistemas Operacionais Modernos***. 3. ed. [S.l.]: Pearson, 2010. v. 1.

WANG, K.; LIN, X.; TANG, W. **Predator - An Experience Guided Configuration Optimizer for Hadoop MapReduce**. *International Conference on Cloud Computing Technology and Science (ICCCTS)*, n. 419-426, p. 8, 2012.

XU, W.; LUO, W.; WOODWARD, N. **Analysis and Optimization of Data Import with Hadoop**. *International Parallel and Distributed Processing Symposium (IPDPS)*, n. 1058-1066, p. 9, 2012.

YUANYUAN, L.; PENG, X. **The Method To Test Linux Software Performance**. *International Conference on Computer and Communication Technologies in Agriculture Engineering (CCTAE)*, n. 420-423, p. 4, 2010.

# 11 APENDICE - CATÁLOGO DE PARÂMETROS

## Catálogo dos Parâmetros de Configurações do Subsistema de Tarefas

Tabela 27 - Parâmetros de Configurações do Subsistema de Tarefas para o Modelo Comum (MC) e Modelo Ajustado (MA).

| **Parâmetro** | **MC** | **MA** | **Observações** |
|---|---|---|---|
| kernel.sched_autogroup_enabled | 1 | 0 | Tarefas *CPU-bound* utilizando pseudo-TTY o impacto é negativo. |
| kernel.sched_cfs_bandwidth_slice_us | 5000 | 5000 | - |
| kernel.sched_child_runs_first | 0 | 0 | - |
| kernel.sched_latency_ns | 60000000 | 100000000 | Observamos que este parâmetro é considerável e relevante. Adotamos estes valores baseando-se inicialmente no referencial teórico. Então, realizamos experimentos com valores superiores e inferiores. Os valores analisados foram 70000000, 100000000 e 140000000. |
| kernel.sched_migration_cost_ns | 500000 | 5000000. | Observamos que este parâmetro é considerável e relevante. Adotamos estes valores baseando-se inicialmente no referencial teórico. Então, realizamos experimentos com valores superiores e inferiores. Os valores analisados foram 200000 (pior caso), 2500000 e 5000000. |
| kernel.sched_min_granularity_ns | 2250000 | 100000 | Observamos que este parâmetro é considerável e relevante. Adotamo este valor baseando-se inicialmente no referencial teórico. Então, realizamos experimentos com valores superiores e inferiores. Os valores analisados foram 50000 (pior caso), 100000 e 250000. |
| kernel.sched_nr_migrate | 32 | 64/128 | Observamos que este parâmetro é considerável e relevante. Adotamo este valor baseando-se inicialmente no referencial teórico. Então, realizamos experimentos com valores superiores e inferiores. Os valores analisados foram 64, 128, 256, 512 e 1024. |
| kernel.sched_rr_timeslice_ms | 25 | 25 | - |
| kernel.sched_rt_period_us | 1000000 | 1000000 | - |
| kernel.sched_rt_runtime_us | 950000 | 950000 | - |
| kernel.sched_schedstats | 1 | 1 | - |
| kernel.sched_shares_window_ns | 10000000 | 10000000 | - |
| kernel.sched_time_avg_ms | 1000 | 1000 | - |
| kernel.sched_tunable_scaling | 1 | 1 | - |
| kernel.sched_wakeup_granularity_ns | 3000000 | A | - |
| fs.file-max | 1024 | 6655634 | Observamos que pode haver impactos negativos com valores iguais ou inferiores a 1024. |
| cgroups | Não aplicado | Aplicado | Observamos que esta abordagem é considerável e relevante. Juntamente com os parâmetros ajustados conseguimos uma redução na latência de 21ms para 3ms (um fator de melhoria de 600%) |
| Afinidade de Processador | Não aplicado | Aplicado | Observamos que esta abordagem é considerável e relevante. A afinidade de processador mitiga o *overhead* da troca de contexto |

## Catálogo dos Parâmetros de Configurações do Subsistema de Memória Virtual

Tabela 28 - Parâmetros de Configurações do Subsistema de Memória Virtual para o Modelo Comum (MC) e Modelo Ajustado (MA).

| **Parâmetro** | **MC** | **MA** | **Observações** |
|---|---|---|---|
| vm.admin_reserve_kbytes | 8192 | 8192 | - |
| vm.block_dump | 0 | 0 | - |
| vm.dirty_background_bytes | 0 | 0 | - |
| vm.dirty_background_ratio | 10 | 50 | Observamos que este parâmetro é considerável e relevante. Com este parâmetro elevamos a quantidade de páginas limpas no sistema. |
| vm.dirty_bytes | 0 | 0 | - |
| vm.dirty_expire_centisecs | 3000 | 0 | Observamos que este parâmetro é considerável e relevante. Com este parâmetro minimizamos a execução da *thread* de kernel para *writeback*. |
| vm.dirty_ratio | 20 | 15 | Com este parâmetro minimizamos a execução da *thread* de kernel para *writeback*. |
| vm.dirty_writeback_centisecs | 500 | 360000 | Observamos que este parâmetro é considerável e relevante. |
| vm.dirtytime_expire_seconds | 3000 | 3000 | - |
| vm.drop_caches | 0 | 0 | - |
| vm.hugepages_treat_as_movable | 0 | 0 | *Huge Pages* não foi contemplado neste trabalho. |
| vm.hugetlb_shm_group | 0 | 0 | *Huge Pages* não foi contemplado neste trabalho. |
| vm.max_map_count | 65530 | 65530 | - |
| vm.min_free_kbytes | 67584 | 67584 | - |
| vm.min_slab_ratio | 5 | 5 | - |
| vm.min_unmapped_ratio | 1 | 1 | - |
| vm.mmap_min_addr | 65536 | 65536 | - |
| vm.nr_hugepages | 0 | 0 | *Huge Pages* não foi contemplado neste trabalho. |
| vm.nr_hugepages_mempolicy | 0 | 0 | *Huge Pages* não foi contemplado neste trabalho. |
| vm.nr_overcommit_hugepages | 0 | 0 | *Huge Pages* não foi contemplado neste trabalho. |
| vm.nr_pdflush_threads | 0 | 0 | - |
| vm.numa_zonelist_order | default | default | - |
| vm.oom_dump_tasks | 1 | 1 | - |
| vm.overcommit_kbytes | 0 | 0 | - |
| vm.overcommit_memory | 0 | 1 | Observamos que este parâmetro é considerável e relevante. Alteramos o comportamento da heurística de reclamação de memória de forma que as aplicações sempre sejam atendias quando ocorrer a reclamação de memória. |
| vm.overcommit_ratio | 50 | 50 | - |
| vm.page-cluster | 3 | 3 | - |
| vm.panic_on_oom | 0 | 0 | - |
| vm.stat_interval | 1 | 1 | - |
| vm.swappiness | 60 | 10 | Observamos que este parâmetro é considerável e relevante. Reduz a propensão a *swap*. |
| vm.user_reserve_kbytes | 124861 | 124861 | - |
| vm.vfs_cache_pressure | 100 | 50 | Consideramos utilizar a metade do valor padrão para minimizar a reclamação de memória de cache do VFS. |
| vm.zone_reclaim_mode | 0 | 0 | - |

## Catálogo dos Parâmetros de Configurações do Subsistema de Arquivos e I/O de Disco

Tabela 29 - Parâmetros de Configurações do Subsistema de Arquivos e I/O de Disco para o Modelo Comum (MC) e Modelo Ajustado (MA).

| **Parâmetro** | **MC** | **MA** | **Observações** |
|---|---|---|---|
| fs.aio-max-nr | 65536 | 65536 | - |
| fs.aio-nr | 0 | 0 | - |
| fs.nr_open | 1048576 | 1048576 | Parâmetro somente para consulta (somente leitura). |
| nr_requests | 128 | 64/256/512/1024 | Observamos que este parâmetro é considerável e relevante. Em nossos experimentos associamos os valores de nr_requests com cada agendador de I/O de disco |
| read_ahead_kb | 128 | 256 | Observamos que este parâmetro é considerável e relevante. Observamos que valores maiores em conjunto com utilização de RAID e leituras sequenciais conseguimos melhor desempenho. |
| I/O Scheduler | Deadline | NOOP/CFQ | Observamos que este parâmetro é considerável e relevante. A escolha correta do agendador de I/O de disco influencia diretamente no desempenho. |
| noatime | Desabilitado | Habilitado | Observamos que este parâmetro é considerável e relevante. Não realiza atualização de tempo de acesso a arquivos. |
| nodiratime | Desabilitado | Habilitado | Observamos que este parâmetro é considerável e relevante. Não realiza atualização de tempo de acesso a diretórios. |
| nobarrier | Desabilitado | Habilitado | Observamos que este parâmetro é considerável e relevante. Desativa recurso de barrier (integridade em caso de falha de energia na controladora). Observamos que a ativação de noatime, nodiratime e nobarrier no sistema de arquivos obtivemos uma melhoria de 10% no *troughput*. |

**Análise da Variância e Desvio Padrão nos Experimentos Realizados**

Nesta análise contemplamos as variações ocorridas entre as execuções dos experimentos quanto ao **Modelo Ajustado**. As tabelas 30 e 32 apresentam os dados em que o desvio padrão está relacionado com o número de migrações de CPU ocorridas.

A tabela 31 apresenta os dados em que o desvio padrão está relacionado com a latência das *threads* com o escalonador CFS.

Na tabela 30 apresentamos o desvio padrão quanto ao tempo de execução, da migração de CPU e também a média da migração de CPU. Nesta tabela anlisamos o parâmetro sched_migration_cost_ns. O número de execuções para valor de sched_migration_cost_ns correspondeu a cinco execuções. O desvio padrão, portanto, se refere à este conjunto de execuções.

Tabela 30 - Desvio Padrão durante as execuções dos experimentos no subsistema de tarefas quanto ao Tempo de Execução e Migração de CPU.

| Parâmetro | Desvio - Tempo Exec. | Desvio - Migr. CPU | Média - Migr. CPU | Valor |
|---|---|---|---|---|
| sched_migration_cost_ns | 0,5 | 100 | 2390 | 2500000 |
| sched_migration_cost_ns | 0,6 | 162 | 2411 | 5000000 |

Tabela 31 - Desvio Padrão durante as execuções dos experimentos no subsistema de tarefas quanto à Latência.

| Parâmetro | Desvio - Latência | Execuções | Valor | Latência Média |
|---|---|---|---|---|
| sched_latency_ns | 0,8 | 5 | 70000000 | 7 ms |
| sched_latency_ns | 0,8 | 5 | 100000000 | 4 ms |
| sched_latency_ns | 0,8 | 5 | 140000000 | 16 ms |
| sched_min_granularity_ns | 0,4 | 5 | 50000 | 11 ms |
| sched_min_granularity_ns | 0,4 | 5 | 100000 | 7 ms |
| sched_min_granularity_ns | 0,4 | 5 | 250000 | 8 ms |

Tabela 32 - Desvio Padrão durante as execuções dos experimentos no subsistema de tarefas quanto à Latência.

| **Parâmetro** | **Desvio - Migração CPU** | **Execuções** | **Valor** | **Média - Migração CPU** |
|---|---|---|---|---|
| sched_nr_migrate | 0,4 | 5 | 64 | 2190 |
| sched_nr_migrate | 0,4 | 5 | 128 | 2104 |
| sched_nr_migrate | 0,5 | 5 | 256 | 2197 |
| sched_nr_migrate | 0,5 | 5 | 512 | 2204 |
| sched_nr_migrate | 0,4 | 5 | 1024 | 2447 |

Tabela 33 - Desvio Padrão durante as execuções dos experimentos no subsistema de I/O de disco analisando os agendadores de I/O de disco NOOP, Deadline e CFQ.

| **Parâmetro** | **Desvio - Troughput** | **Execuções** | **Valor** | **Média - Troughput** |
|---|---|---|---|---|
| NOOP | 0,5 | 5 | - | 385,04 MB/S |
| Deadline | 0,9 | 5 | - | 235,3 MB/S |
| CFQ | 0,2 | 5 | - | 227,8 MB/S |

Tabela 34 - Desvio Padrão durante as execuções dos experimentos no subsistema de I/O de disco analisando os agendadores de I/O de disco NOOP, Deadline e CFQ quanto às variações do parâmetro nr_requests.

| **Agendador** | **nr_requests** | **Execuções** | **Desvio - *Throughput*** | **Média - *Throughput*** |
|---|---|---|---|---|
| NOOP | 64 | 5 | 1,81 MB/S | 468,9 MB/S |
| Deadline | 64 | 5 | 1,21 MB/S | 708,8 MB/S |
| CFQ | 64 | 5 | 2,6 MB/S | 337,8 MB/S |
| NOOP | 128 | 5 | 1,18 MB/S | 325,7 MB/S |
| Deadline | 128 | 5 | 4,58 MB/S | 297,4 MB/S |
| CFQ | 128 | 5 | 3,15 MB/S | 296,8 MB/S |
| NOOP | 256 | 5 | 1,43 MB/S | 230,7 MB/S |
| Deadline | 256 | 5 | 12,0 MB/S | 411,9 MB/S |
| CFQ | 256 | 5 | 6,58 MB/S | 415,3 MB/S |
| NOOP | 512 | 5 | 8,83 MB/S | 312,7 MB/S |
| Deadline | 512 | 5 | 8,77 MB/S | 438,4 MB/S |
| CFQ | 512 | 5 | 11,75 MB/S | 448,7 MB/S |
| NOOP | 1024 | 5 | 10,92 MB/S | 332,5 MB/S |
| Deadline | 1024 | 5 | 7,61 MB/S | 362,5 MB/S |
| CFQ | 1024 | 5 | 2,24 MB/S | 639,4 MB/S |
| NOOP | 2048 | 5 | 2,45 MB/S | 221,7 MB/S |
| Deadline | 2048 | 5 | 9,71 MB/S | 227,6 MB/S |
| CFQ | 2048 | 5 | 7,35 MB/S | 209,5 MB/S |
| NOOP | 4096 | 5 | 50,08 MB/S | 232,8 MB/S |
| Deadline | 4096 | 5 | 21,28 MB/S | 170,3 MB/S |
| CFQ | 4096 | 5 | 13,71 MB/S | 234,9 MB/S |

**Considerações Sobre Outros Parâmetros de Configurações**

As tabelas 35, 36 e 37 apresentam os parâmetros que foram analisados, mas não foram selecionados, seja por não estarem relacionados com o objetivo deste trabalho ou não possuirem impacto no desempenho. Estas duas premissas foram corroboradas por meio das fontes (referencial teórico, trabalhos relacionados e outros) analisadas.

Tabela 35 - Parâmetros de Configurações Analisados e Não Contemplados no Subsistema de Tarefas.

| **Parâmetro** | **Subsistema** | **Observações** |
|---|---|---|
| kernel.modules_disabled | Tarefas/Kernel | - |
| kernel.msgmax | Tarefas/Kernel | - |
| kernel.msgmnb | Tarefas/Kernel | - |
| kernel.msgmni | Tarefas/Kernel | - |
| kernel.ngroups_max | Tarefas/Kernel | - |
| kernel.numa_balancing | Tarefas/Kernel | Arquitetura NUMA. |
| kernel.numa_balancing_migrate_deferred | Tarefas/Kernel | Arquitetura NUMA. |
| kernel.numa_balancing_scan_delay_ms | Tarefas/Kernel | Arquitetura NUMA. |
| kernel.numa_balancing_scan_period_max_ms | Tarefas/Kernel | Arquitetura NUMA. |
| kernel.numa_balancing_scan_period_min_ms | Tarefas/Kernel | Arquitetura NUMA. |
| kernel.numa_balancing_scan_size_mb | Tarefas/Kernel | Arquitetura NUMA. |
| kernel.numa_balancing_settle_count | Tarefas/Kernel | Arquitetura NUMA. |
| kernel.randomize_va_space | Tarefas/Kernel | Segurança. Randomização completa do espaço de endereços. |
| kernel.sched_cfs_bandwidth_slice_us | Tarefas/Kernel | - |
| kernel.sched_shares_window_ns | Tarefas/Kernel | - |
| kernel.sched_shares_window_ns | Tarefas/Kernel | - |
| kernel.sched_time_avg_ms | Tarefas/Kernel | - |
| kernel.sched_tunable_scaling | Tarefas/Kernel | - |
| kernel.shmall | Tarefas/Kernel | Memória compartilhada. |
| kernel.shmmax | Tarefas/Kernel | Memória compartilhada. |
| kernel.shmmni | Tarefas/Kernel | Memória compartilhada. |
| kernel.threads-max | Tarefas/Kernel | - |

Tabela 36 - Parâmetros de Configurações Analisados e Não Contemplados no Subsistema de Memória Virtual.

| **Parâmetro** | **Subsistema** | **Observações** |
|---|---|---|
| vm.extfrag_threshold | Memória Virtual | - |
| vm.hugepages_treat_as_movable | Memória Virtual | Páginas grandes (ou gigantes). |
| vm.hugetlb_shm_group | Memória Virtual | - |
| vm.lowmem_reserve_ratio | Memória Virtual | - |
| vm.max_map_count | Memória Virtual | - |
| vm.memory_failure_early_kill | Memória Virtual | - |
| vm.memory_failure_recovery | Memória Virtual | - |
| vm.min_free_kbytes | Memória Virtual | - |
| vm.min_slab_ratio | Memória Virtual | - |
| vm.nr_hugepages | Memória Virtual | Páginas grandes (ou gigantes). |
| vm.nr_hugepages_mempolicy | Memória Virtual | Páginas grandes (ou gigantes). |
| vm.nr_overcommit_hugepages | Memória Virtual | Páginas grandes (ou gigantes). |
| vm.nr_pdflush_threads | Memória Virtual | - |
| vm.page-cluster | Memória Virtual | - |
| vm.user_reserve_kbytes | Memória Virtual | - |

Tabela 37 - Parâmetros de Configurações Analisados e Não Contemplados no Subsistema de Arquivos e I/O de Disco.

| **Parâmetro** | **Subsistema** | **Observações** |
|---|---|---|
| add_random | /sys/block/sda/queue | - |
| discard_granularity | /sys/block/sda/queue | - |
| discard_max_bytes | /sys/block/sda/queue | - |
| discard_zeroes_data | /sys/block/sda/queue | - |
| write_same_max_bytes | /sys/block/sda/queue | - |