PROGRAMA DE EDUCAÇÃO CONTINUADA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO

Luis Gustavo Pariz Campos

ARQUITETURA DE ARMAZENAMENTO ON-PREMISE APLICADA À MANIPULAÇÃO DE ARQUIVOS PEQUENOS EM HDFS

São Paulo

2016

Luis Gustavo Pariz Campos

ARQUITETURA DE ARMAZENAMENTO ON-PREMISE APLICADA À MANIPULAÇÃO DE ARQUIVOS PEQUENOS EM HDFS

Monografia apresentada ao PECE – Programa de Educação Continuada da Escola Politécnica da Universidade de São Paulo para obtenção do título de Especialista pelo programa de Big Data: Inteligência na Gestão dos Dados.

Orientador: Profa. Dr. Luiz Sérgio de Souza

São Paulo

2016

Dedico este trabalho à minha família, aos professores e amigos que estão sempre próximos para compartilhar dificuldades e êxitos.

"Há homens que lutam um dia e são bons.

Há outros que lutam um ano e são melhores.

Há os que lutam muitos anos e são muito bons.

Porém, há os que lutam toda a vida.

Esses são os imprescindíveis."

Bertolt Brecht

RESUMO

O HDFS foi concebido para manipular arquivos grandes. Contudo, se observa que em cenários com grande volume de dados oriundos de redes sociais ou aplicações do tipo near real-time, normalmente, são geradas enxurradas de arquivos com tamanho menor que a blocagem padrão do HDFS. Isso leva, a um uso não otimizado do HDFS. Por outro lado, a literatura apresenta soluções que visam melhorar o desempenho desse sistema de arquivos. Entretanto, encontrar e selecionar soluções adequadas exige tempo, o que normalmente falta aos profissionais devido à grande carga de trabalho diário. Assim, para auxiliar os profissionais responsáveis em desenhar soluções de software e infraestrutura, essa monografia reúne, numa única fonte, informações necessárias que auxiliem o profissional a estruturar o HDFS de forma mais eficiente, com melhor desempenho nas operações de leitura e escrita. Os estudos, que foram base para a elaboração dessa monografia, melhoram o desempenho do HDFS em até 7x nas operações de escrita e duplicação em instruções de leitura.

Palavras-chave: HDFS. Hadoop. Arquivos pequenos. Armazenamento Heterogêneo

ABSTRACT

The HDFS has been designed to handle large files. However, if notes that in scenarios with large volume of data from social networks or applications of the near real-time type usually are storms generated files with size less than blocking standard of HDFS. This leads to a non-optimized use of HDFS. On the other hand, the literature presents solutions to improve the performance of this file system. However, find and select appropriate solutions requires time, which typically lack the professionals due to the large supply of daily work. Therefore, to help the professionals responsible to design software and infrastructure solutions, this monograph brings together in a single source information necessary to help the professional to structure the HDFS more effectively, with better performance in reading and writing. The studies, which were basis for this monograph, improve the performance of the HDFS by up to 7 x in write operations and duplicate in reading instructions.

Keywords: HDFS. Hadoop. Small files. Heterogeneous Storage

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

CAPEX	Capital Expenditure
CIFS	Common Internet File System
DAS	Direct Attached Storage
FC	Fibre Channel
FCIP	Fibre Channel over TCP/IP
FCP	Fibre Channel Protocol
GFS	Google’s Distributed File System
HIPI	Hadoop Image Processing Interface
HMPI	Hadoop Multimedia Processing Interface
HPC	High-Performance Computing
ISCSI	Internet Small Computer System Interface
JAR	Java Archive
LAN	Local Area Network
NAS	Network Attached Storage
NFS	Network File System
POSIX	Portable Operating System Interface
RAID	Redundant Array of Independent Disks
SAN	Storage Area Network
SMB	Server Message Block
WAN	Wide Area Network
WORM	Write-Once-Read-Many

SUMÁRIO

1 INTRODUÇÃO

O início do planejamento de construção de um ambiente computadorizado necessita de uma arquitetura bem definida para se obter êxito. A arquitetura pode ser compreendida como a forma que se dispõem as partes ou os elementos em um determinado cenário. Ainda assim, pode-se observar que a arquitetura relaciona normas, materiais e técnicas, especificações e formas de execução.

Dentro do tempo necessário para a definição de uma arquitetura diversos pontos importantes devem ser analisados, cada cenário merece atenção às suas peculiaridades e limitações, pois não existe uma chave mestra, que solucione todas as necessidades.

Os dados gerados por indivíduos ou por máquinas devem ser armazenados e recuperados de forma rápida e fácil. No âmbito da computação, as soluções de armazenamento precisam ser desenhadas para atender à estes requisitos. Os dados podem persistir em um cenário caseiro, por exemplo: em cartões de memória, CDs, DVDs e discos rígidos. Já no cenário corporativo, são diversas as formas que se pode encontrar para armazenar dados, como exemplos: soluções DAS (*Direct-Attached Storage*), SAN (*Storage-Area Network*), NAS (*Network-Attached Storage*), fitas entre outras tecnologias.

A transformação do modelo de arquitetura de armazenamento, tem-se dado pela descentralização dos equipamentos que mantém os dados acessíveis. Historicamente, os servidores usavam um desenho, tipicamente, interno de persistência. Com o tempo, esse modelo sofreu alterações e os dados passaram a não mais serem armazenados internamente, provendo a possibilidade de gerenciamento independente dos dados. (GNANASUNDARAM; SHRIVASTAVA, 2012)

A evolução dos meios de armazenamento está permitindo acompanhar o avanço do volume de dados gerados. Essa geração não apresenta comportamento linear, visto que 90% de todos dados de hoje foram criados somente nos últimos dois anos. O aumento da capacidade de processamento, em conjunto com o desenvolvimento de equipamentos periféricos portáteis, têm trazido um cenário favorável à Internet das

Coisas. Esse conjunto de tecnologias que conecta uma infinidade de itens, gera dados a todo instante e se torna um importante impulsionador para *Big Data*.

A primeira fase de um processo *Big Data* está na coleta dos dados. Uma vez que esses dados são coletados, eles precisam ser armazenados em uma determinada área local, chamado de modelo on-premise, para se extrair informações. (TAURION, 2013). No contexto de tecnologias *Big Data*, o framework Hadoop foi concebido para processamento distribuído dos dados. O HDFS (Hadoop *Distributed File System*) é o sistema de arquivo que dá suporte ao Hadoop para o armazenamento distribuído, performático, capaz de guardar volumetrias significativamente grandes.

1.1 Motivação e Justificativa

Este trabalho iniciou-se a partir da observação do cotidiano dos arquitetos de soluções de armazenamento, que com a crescente avalanche de dados oriundos do cenário de *Big Data*, têm encontrado dificuldades para prover desempenho necessário ao negócio.

O estudo foi então embasado na avaliação dos problemas encontrados para realização de armazenamento de dados provenientes de utilitários do Hadoop. Observou-se que o HDFS foi projetado, inicialmente, para ser utilizado com arquivos grandes (maiores que 64 MB), entretanto, têm-se notado que na verdade são os pequenos arquivos (menores que 64 MB) os quais estão sendo mais persistidos e que de certa forma prejudicam o desempenho das aplicações.

A resultante de um *design* de armazenamento eficaz é peça fundamental para obter êxito na aplicação. Alguns itens são essenciais nessa fase e devem ser levados em consideração:

- Tempo de resposta desejado pela aplicação;
- Área necessária para atender o volume inicial e crescimento esperado;
- Nível de proteção dos dados (integridade dos dados);
- Forma de acesso ao repositório;

A busca por tecnologias com melhor desempenho, aliadas à economia, tem levado a forma como os dados são armazenados sofrerem mudanças. Nesse cenário de

transformação, a utilização do HDFS para persistir os dados, principalmente, em ambientes com grande quantidade dados, tem sido a forma mais escolhida. Acolhendo a essa tendência e notando a dificuldade encontrada por profissionais da área de arquitetura, pretende-se gerar estudos que possam servir como um guia para resolução dos problemas encontrados no HDFS com arquivos pequenos e as limitações do modelo de armazenamento.

1.2 Objetivo

O objetivo deste trabalho é apresentar soluções de concatenação dos arquivos e armazenamento heterogêneo encontradas na literatura para melhorar e eficiência nas operações de leitura e escrita no HDFS, considerando uma carga massiva de arquivos pequenos.

1.3 Contribuição

Como contribuição busca-se apresentar num único material informações encontradas em diferentes pesquisas que apresentam soluções para melhoria do desempenho do HDFS com quantidade massiva de arquivos de tamanho reduzido (menores que 64 MB). O trabalho ainda poderá servir à outras pesquisas relacionadas ao armazenamento em sistemas de arquivos distribuídos.

1.4 Metodologia

O trabalho iniciou-se com a observação da dificuldade encontrada por arquitetos e outros profissionais de TI, em desenhar estruturas destinadas ao HDFS que possuam bom desempenho com arquivos pequenos. Deste modo, com a necessidade de trazer melhorias aos processos e desempenho no armazenamento de arquivos menores que a blocagem padrão do HDFS, foram realizadas pesquisas bibliográficas no anseio de buscar soluções que trouxessem uma forma possível de desenhar esse ambiente. Na pesquisa estão contidos elementos de artigos que auxiliam na

Após a identificação deste problema, foi iniciada a busca bibliográfica por soluções através de livros, fóruns e artigos técnicos relacionados à arquitetura de

armazenamento que abortassem essa deficiência. Entretanto, não foram encontrados materiais que unissem soluções completas, fim-a-fim, para manipulação de arquivos pequenos, aliadas à persistência inteligente dos blocos.

Foi encontrado entre os artigos elencados, um conjunto de soluções que propiciam ao HDFS desempenho para atender às necessidades de ambientes *Big Data,* com essas características. As propostas foram alinhadas, ao passo que o resultado deste trabalho foi um guia, que parte do entendimento da necessidade do negócio, ou seja, o *throughput* esperado pelo sistema/aplicação, chegando à adequação da manipulação e persistência dos arquivos, empregando técnicas recomendadas pelas referências elegidas.

1.5 Estruturação

Este trabalho foi divido da seguinte forma: no Capítulo 2 têm-se os preceitos teóricos desta monografia, que abordam a arquitetura de armazenamento, o sistema de arquivos distribuídos do *framework* Hadoop e sua utilização em cenários de grande volume de dados. Estão também presentes nessa seção, os trabalhos correlatos que nortearam o desenvolvimento desta monografia.

No Capítulo 3, apresenta-se o desenvolvimento do trabalho proposto, através de exemplos de soluções de armazenamento e técnicas usadas para melhoria do desempenho que estão surgindo para aperfeiçoar a experiência com *Big Data*. Finalmente, no Capítulo 4, têm-se as considerações finais, a conclusão e propostas aos trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nesse capítulo aborda-se a base conceitual necessária ao desenvolvimento deste trabalho. Na seção 2.1 estão as definições iniciais de arquitetura, na seção 2.2 está a evolução da arquitetura de armazenamento e tecnologias utilizadas atualmente, na seção 2.3 está o *framework* Hadoop e seus principais aplicativos. Ao final, na seção 2.4 está o detalhamento do HDFS.

2.1 Arquitetura

De acordo com Michaelis (2016) arquitetura pode ser entendida como a forma que se dispõem as partes ou os elementos dado um cenário. Deste modo, pode-se observar que a arquitetura relaciona normas, materiais e técnicas, especificações e formas de execução.

No âmbito computacional, focado em dados, a arquitetura de armazenamento descreve como os dados são processados, persistidos e servem ao negócio, em geral. Ainda especifica os critérios para as operações de processamento de dados, deste modo, é possível controlar o fluxo das informações.

A responsabilidade do arquiteto nesse segmento, é garantir que os dados serão acessados pelas aplicações com eficiência, por meio de especificações acertadas de acordo com a necessidade do negócio. A melhoria dos resultados acompanha todo o processo, o trabalho de agregar melhores rendimentos aos equipamentos de armazenamento estão no dia-a-dia deste profissional. Todas essas ações são realizadas usando-se da divisão da arquitetura em: conceitual, físico e lógico.

2.1.1 Arquitetura Conceitual de Dados

Essa forma de arquitetura tem como objetivo confeccionar um modelo conceitual de armazenamento dos dados. Traz uma visão de alto nível do ambiente, o qual dá assistência às necessidades do negócio de uma organização, norteando as decisões sobre as tecnologia dispostas. Nesse modelo, o destaque é para relações de negócio da empresa como um todo, descartando-se assim, inicialmente,

limitações tecnológicas, pois estas serão tratadas nas próximas formas, mais amadurecidas nesse aspecto. (KLUG; TSICHRITZIS, 1975)

2.1.2 Arquitetura Lógica de Dados

Na arquitetura lógica de dados, são descritos com detalhes as propriedades e os relacionamentos de cada um dos grupos de dados relacionados ao negócio. Nessa forma de arquitetura, têm-se como produto: estruturas normatizadas, relacionamento entre os dados, além de um modelo organizacional de gerenciamento. (KLUG; TSICHRITZIS, 1975)

2.1.3 Arquitetura Física de Dados

A arquitetura física de dados está diretamente ligada ao meio físico que os dados serão persistidos, concentrando as atenções na composição de elementos reais e tangíveis a serem utilizados. É nessa fase que são definidas no detalhe as especificações técnicas e tecnologias responsáveis pelo correto funcionamento das aplicações e, por consequência, suporte necessário ao negócio. (KLUG; TSICHRITZIS, 1975)

2.2 Evolução da arquitetura de armazenamento

A evolução da arquitetura de armazenamento, está na descentralização do modelo de acesso aos dados. Historicamente, os servidores usavam um modelo conhecido como DAS (*Direct Attached Storage*), método local de se conectar aos dispositivos de armazenamento. Na arquitetura direta de acesso, cada servidor possui um número restrito de dispositivos exclusivos e as tarefas administrativas, como manutenções ou crescimento de capacidade, ocasionavam indisponibilidade das informações. (GNANASUNDARAM; SHRIVASTAVA, 2012)

Todavia a necessidade por gerenciar os dados de forma não exclusiva pelo servidor hospedeiro, fez este cenário centralizado no servidor (*Server-Centric* Fig.1) ser substituído por outro modelo centrado em informações (*Information-Centric* Fig. 2).

Figura 1 Modelo centralizado no servidor

Fonte: Gnanasundaram e Shrivastava, 2012.

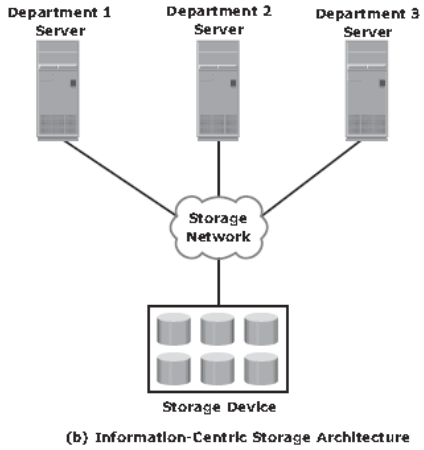

Figura 2 Modelo centralizado na informação

. Fonte: Gnanasundaram e Shrivastava, 2012

Através desta mudança, o modelo sofreu alterações e os dados passaram a não mais serem armazenados internamente, possibilitando através de uma rede de armazenamento, acesso por diversos servidores ao mesmo tempo. Além disso, a evolução da arquitetura de armazenamento trouxe o gerenciamento independente dos dados, deste modo, as atividades como: crescimento de capacidade de armazenamento, manutenções em servidores e migrações de ambientes, passaram a não mais interromper as aplicações, protegendo os negócios da instituição. (GNANASUNDARAM; SHRIVASTAVA, 2012)

2.2.1 Tecnologias de Armazenamento

As tecnologias de armazenamento de dados são nomeadas de acordo com as formas pelas quais são conectados os dispositivos de armazenamento de dados aos sistemas computacionais e pelo tipo de informação que é trocada entre eles. As principais tecnologias de armazenamento existentes são DAS (*Direct Attached Storage*), NAS (*Network Attached Storage*) e SAN (*Storage Area Network*). Como indicado anteriormente, o modelo DAS tem o acesso através de um barramento local ou outro meio físico aos dispositivos de armazenamento. Já o modelo NAS permite acesso através de redes LAN e WAN por meio de protocolos como NFS (*Network File System*), SMB (*Server Message Block*) e CIFS (*Common Internet File System*). Por último, o modelo SAN que provê uma rede dedicada de transferência de dados

em baixo nível, similar às instruções internas em discos, como SCSI. (GNANASUNDARAM; SHRIVASTAVA, 2012)

Para dividir as tecnologias de armazenamento de dados, será utilizado o modelo desenvolvido pela SNIA. Criada em 1997, a SNIA (*Storage Network Industry Association*) é uma organização global sem fins lucrativos, constituída por empresas associadas, abrangendo o mercado global de armazenamento de dados, cuja missão é liderar a indústria de armazenamento mundial no desenvolvimento e promoção de padrões, tecnologias e serviços educacionais para capacitar as organizações na gestão da informação. (SNIA, 2016)

Através do modelo criado pela SNIA (Fig. 3), o qual é chamado de Modelo de Armazenamento Compartilhado de Dados (*Shared Storage Model*), pode-se apresentar uma estrutura genérica para arquitetura de armazenamento de dados compartilhados, relacionando os serviços com as formas de acesso aos dados. Por meio deste modelo, se torna viável mapear o cenário presente de armazenamento de dados para as soluções propostas, ajuda a esclarecer o que as tecnologias estão endereçando e também cria uma base de conhecimento para atender futuras necessidades da aplicação.

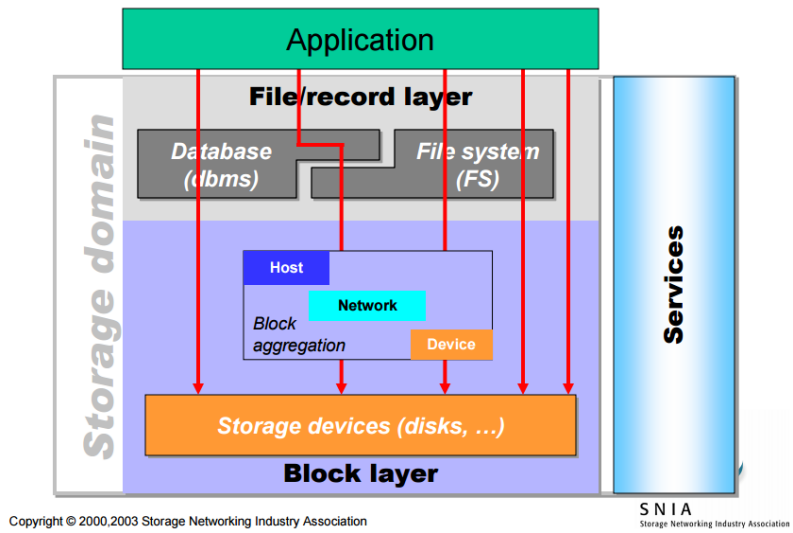

Figura 3 Modelo SNIA de armazenamento de dados compartilhado.

Fonte: SNIA, 2016.

Ao analisar a Fig. 3, percebe-se que o modelo proposto pela SNIA estabelece uma aliança entre a aplicação, que é executada no âmbito computacional, ou seja, nos servidores e o domínio de armazenamento de dados. Nesse modelo de arquitetura de dados, pode-se subdividir o domínio de armazenamento em duas camadas:

- **Camada de Arquivo/Registro (*File/ Record Layer*) -** é a interface entre o nível mais alto de aplicações e os recursos de armazenamento**,** representada pela sequência de bytes de informações que formam registros ou arquivos.
- **Camada de Blocos (*Block Layer*) -** é a faixa de baixo nível do armazenamento onde os blocos de dados são persistidos ou lidos no *hardware*, comumente chamado de *Storage*.

Através do Modelo de Armazenamento Compartilhado de Dados, proposto pela SNIA, pode-se tratar das diferentes formas de armazenamento, listando as principais características das tecnologias DAS, NAS e SAN.

2.2.1.1 *Direct Attached Storage* (DAS)

A tecnologia de armazenamento compreendia como DAS, é a forma de persistir os dados mais simples, pela ausência de uma rede de comunicação entre o servidor e o periférico de armazenamento, os discos são internos ou estão em um gabinete conectado diretamente ao servidor que faz a gerência dos dados. A comunicação nesse modelo de arquitetura é feita, usualmente, pela tecnologia SCSI (*Small Computer System Interface*) paralelo. (GNANASUNDARAM; SHRIVASTAVA, 2012)

Nesse modelo de entrega de volumetria, o canal de comunicação físico possui limitações restritas de distância e dificuldade para entregar alta carga de dados. Além disso, outro fator que restringe o uso deste modelo em cenários empresariais, é a limitação da quantidade de discos e indisponibilidade em caso de manutenção.

DAS requer um investimento inicial muito inferior se comparado à outras tecnologias como SAN, por exemplo. Tornando-se uma arquitetura fácil e rápida de ser implantada, onde a configuração e gerenciamento dos blocos são realizados pelo sistema operacional do servidor. (GNANASUNDARAM; SHRIVASTAVA, 2012)

2.2.1.2 *Network Attached Storage* (NAS)

Network Attached Storage provê o compartilhamento de arquivos de forma flexível, por meio de uma rede de comunicação não exclusiva à *Storage* que une os servidores clientes aos dispositivos de armazenamento, permitindo alcançar longas distancias, com massiva quantidade de usuários, benefício de altas velocidades de transmissão, alta disponibilidade e consolidação de armazenamento. (PRESTON, 2002)

A essência do modelo de arquitetura NAS, segundo as definições da SNIA, está em dispositivos ligados à redes LAN (*Local Area Network*) e/ou WAN *(Wide Area Network*) que trafegam os arquivos através de protocolos de compartilhamento de arquivos na rede. No cenário atual, os principais protocolos de compartilhamento de arquivos são CIFS/SMB e NFS, destinados aos sistemas operacionais Microsoft Windows e Unix/Linux, respectivamente.

Os equipamentos de armazenamentos chamados de NAS usam seu próprio sistema operacional, hardware integrado e componentes de software para atender às necessidades específicas e executar de forma otimizada o compartilhamento de arquivos. Seu sistema operacional é desenvolvido para instruções I/O de arquivo e, portanto, executa estas instruções de máquina melhor do que um servidor ou PC (*Personal Computer*) de uso geral. Como resultado, um dispositivo NAS pode servir a mais clientes do que servidores de uso geral, aprovisionando consolidação de armazenamento e melhor tempo de resposta. (GNANASUNDARAM; SHRIVASTAVA, 2012)

2.2.1.3 *Storage Area Network* (SAN)

A arquitetura de armazenamento conhecida como SAN é de modo geral um modelo composto por uma rede dedicada ao tráfego de dados, geralmente, a nível de bloco que permite que diversos servidores tenham acesso à volumetrias externas de modo rápido, com alta disponibilidade e segurança. A entidade de armazenamento que está disponível externamente não é limitada em espaço como em uma arquitetura DAS, onde o dispositivo é interno e por essa razão nesse modelo são mais fáceis de manusear quando necessário e trazem facilidades para o compartilhamento por

vários servidores. Em um cenário SAN, o dispositivo central é o equipamento responsável pela gerência e disponibilização do dados, o qual é comumente chamado de *Storage*. Através da figura do *Storage*, ocorre a centralização das operações de armazenamento e o gerenciamento dos dados passa a ser tarefa não mais exclusiva do servidor. São essas as razões principais pela popularização e utilização em grande escala das grandes empresas pelo modelo SAN. (GNANASUNDARAM; SHRIVASTAVA, 2012)

A idealização do modelo SAN iniciou-se com a necessidade de agrupar uma quantidade elevada de discos, afim de atender à crescente necessidade por volumetria ao passo que a proteção das informações também ganhava importância, através da organização dos discos em RAID (*Redundant Array of Independent Disks*). Outro fator que impulsionou o surgimento desta arquitetura foi o desenvolvimento do padrão FC (*Fibre Channel*), que é um protocolo de transferência de dados por meio de fibra ótica, que propicia a transmissão em altas velocidades como 16 Gb/s ou superior. (SNIA, 2016)

Segundo a definição proposta pela SNIA sobre SAN, qualquer tipo de rede dedicada ao uso de armazenamento pode ser usada, no entanto, atualmente, as redes baseadas em *Fibre Channel* e *Gigabit Ethernet* são a mais comuns. Nesse cenário aparecem as redes mais utilizadas: FCP (*Fibre Channel Protocol*), FCIP (*Fibre Channel over TCP/IP*) e ISCSI (*Internet Small Computer System Interface*).

No cenário atual, os dados das empresas são os seus ativos mais valiosos e, *Big Data* aumenta as exigências das tecnologias de armazenamento presentes, com sua forma exponencial de crescimento de dados estruturados e não-estruturados, além da necessidade por tempos de resposta cada vez menores, essências a determinadas áreas de negócios. (TAURION, 2013) É sobre esse panorama que surge o *framework* Hadoop para proporcionar maior capacidade de processamento e inteligência em armazenamento distribuído.

2.3 *Framework* Hadoop

Criado por Doug Cutting, o Hadoop foi originado a partir do Apache Nutch, uma solução *open source* de pesquisas na web, o qual fazia parte do projeto Apache Lucene, uma biblioteca de pesquisas, amplamente, utilizado. (WHITE, 2012)

O projeto Nutch teve início em 2002, sendo utilizado como uma ferramenta de pesquisa rápida e rastreador web (*web crawler*). Entretanto, foi percebido que essa arquitetura não seria capaz de crescer à escala de bilhões de páginas. Em 2003, por meio da publicação de um artigo a Google apresentou a arquitetura do Google's *Distributed File System*, chamado de GFS e, utilizado no ambiente produtivo desta empresa. Nessa publicação, estavam as principais soluções dos problemas encontrados pelo projeto Nutch com a gerência de armazenamento de arquivos grandes. Em 2004, o Google apresentou ao mundo o MapReduce. Já entre os anos de 2005 e 2006 e, com a chegada de Doug Cutting ao Yahoo, o projeto passou a ser chamado de Hadoop. (WHITE, 2012)

De acordo com a organização Apache, o Hadoop é um *framework* que permite o processamento distribuído de larga escala de dados através de computação clusterizada, usando modelos simplificados de programação que podem ser escalonados à milhares de servidores.

Na versão abordada (2.x), o framework Hadoop é composto pelos módulos:

- **Hadoop Commom** - o núcleo da estrutura, pois fornece serviços essenciais e processos básicos, como a abstração do sistema operacional subjacente e seu sistema de arquivos. O Hadoop Common também contém os arquivos JAR (*Java Archive*) necessários, os scripts para iniciar o Hadoop, código-fonte, documentações, além de uma sessão dedicada a contribuição de outros subprojetos do Hadoop.
- **Hadoop Yarn** - uma ferramenta responsável pelo gerenciamento dos recursos computacionais em cluster e agendamento destes recursos. Na estrutura do Yarn possui um Gerenciador de Recursos (*Resource Manager*) global e um Mestre por aplicação (*Application Master*). O Gerenciador de Recursos e o Gerenciador do Nó (*Node Manager*) constituem a estrutura

computacional. O Gerenciador de Recursos é a autoridade final que arbitra recursos entre todas as aplicações no sistema. Já o Gerenciador do Nó é responsável pela estrutura por máquina e monitora o uso de recursos como CPU, memória, utilização em disco e utilização de rede, narrando os ao Gerenciador de Recursos. A figura do Gerenciador de Aplicações tem a função de orquestrar as tarefas submetidas, negociar o contêiner para executar um determinado aplicativo e prover os serviços necessários para reiniciar um Mestre em caso de um contêiner apresentar falha.

- **Hadoop MapReduce** – é um modelo computacional distribuído, que utiliza-se de funções oriundas da programação funcional. O ambiente compreendido por MapReduce provê aos usuários uma experiência sofisticada na gerência de mapeamento e redução em tarefas de um *cluster* Hadoop. (VENNER, 2009) O fluxo das operações desta solução inicia-se com a leitura de uma entrada, onde o leitor divide os dados em blocos para a próxima fase. Na função de mapeamento, os blocos obtidos através do leitor recebem uma ou mais combinações de chave e valor. Com o resultado da função de mapeamento um redutor é designado a efetuar a partição, deste modo nesta fase ocorre a distribuição da carga de processamento entre os nós. Após a tarefa de *map* concluída, o resultado é comparado com o valores definidos e direcionado à função *reduce*, que resume os itens encontrados gerando uma única saída ou somatório de ocorrências que serão persistidos em um sistema de armazenamento, como o HDFS, por exemplo.
- **Hadoop *Distributed File System* (HDFS) –** um sistema de arquivos distribuído adotado pelo f*ramework* Hadoop para persistir grande quantidade de dados, de forma rápida, segura e escalável a milhares de nós. Através deste sistema é possível conectar servidores comuns, conhecidos na estrutura do HDFS como nós, contidos em clusters onde os blocos de dados são distribuídos e assegurados por meio de replicações. Deste modo, ocorre o acesso e armazenamento dos blocos de dados como um sistema de arquivos contínuo que usa o modelo de processamento MapReduce. Esta solução de armazenamento assemelha-se as demais formas de

compartilhamento de arquivos já presentes no mercado, porém suporta algumas diferenças importantes, uma destas está na arquitetura de leitura e escrita, pois o HDFS utiliza o modelo WORM (*Write-Once-Read-Many*) que facilita as requisições do domínio de simultaneidade, simplifica a persistência de dados e possibilita o acesso de alto rendimento aos sistemas de arquivos

2.4 HDFS

Como o foco deste trabalho está na arquitetura do Hadoop *Distributed File System* aplicada à ambientes locais (leia-se *on-premise*), nessa seção é apresentada uma descrição mais detalha deste subprojeto da Apache *Software Foundation* e solução padrão de persistência de dados do Hadoop.

Criado a partir da necessidade de atender às exigências de volumes de dados que tendem ao crescimento exponencial, na ordem de *terabytes* e *petabytes*, o HDFS aparece como uma solução de persistência de dados em *Big Data*, pelas principais características de:

- Proteção contra falhas pela detecção de erros e aplicação de recuperação rápida e automática;
- Acesso aos dados através do fluxo Hadoop MapReduce;
- Modelo de distribuição de arquivos simultâneos simples e robusto;
- Lógica de processamento orientada aos dados;
- Exemplo de sistema do tipo POSIX (*Portable Operating System Interface*) que garante a portabilidade entre sistemas operacionais e *hardware* heterogêneos;
- Escalabilidade para registrar e tratar de modo confiável larga quantidade de dados
- Redução dos gastos com CAPEX (*Capital Expenditure*) por ser uma solução econômica;
- Eficácia na distribuição de dados e processamento em paralelo que ocorre nos nós em que os dados estão armazenados;

- Segurança e confiabilidade pela replicação automática de várias cópias dos dados em um cenário à prova de falhas.

O HDFS possui uma arquitetura do tipo mestre/escravo. Em um *cluster* HDFS têm-se um único NameNode, responsável pela gerência do sistema de arquivo que tem a figura de mestre sobre os nós do tipo escravo, chamados de DataNodes. O NameNode ainda executa operações de *namespace* do sistema de arquivo como abertura, fechamento e renomeação de arquivos e diretórios, além de determina o mapeamento dos blocos entre os DataNodes. Já a figura do DataNode é responsável por garantir a leitura e escrita das solicitações dos clientes, ainda assim, tem a ação de criação de blocos de dados, bem como a exclusão destes duplicados no nó. A seguir, pode-se ver na Fig. 4 como são distribuídos esses elementos. (BORTHAKUR, 2015)

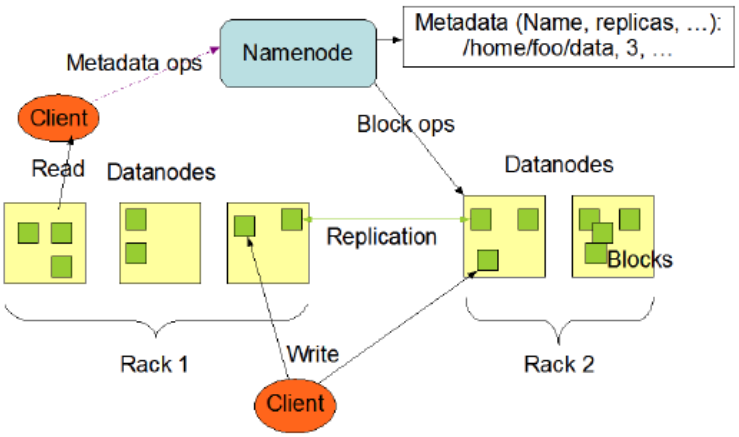

Figura 4 Arquitetura do HDFS.

Fonte: Borthakur, 2008.

Os elementos desta arquitetura, o NameNode e DataNode são figuras de software projetadas para rodar em hardwares de *commodities*. Estes servidores geralmente executam sistemas operacionais oriundos de distribuições Linux. HDFS é construído usando a linguagem Java, qualquer servidor que possua suporte à Java pode realizar o papel de NameNode ou DataNode. Através da linguagem Java, que é altamente portátil, o HDFS pode ser implantado em uma ampla faixa de máquinas.

Faz parte da figura do NameNode a responsabilidade por manter o *namespace* do sistema de arquivo. Qualquer alteração no *namespace* ou nas propriedades do

sistema de arquivo, estas são registradas pelo NameNode. Nas configurações presentes no NameNode, estão também as opções de segurança, onde um aplicativo pode especificar o número de réplicas de um arquivo que deve ser mantido pelo HDFS. O número de cópias de um arquivo, nessa estrutura é chamado de fator de replicação de arquivo. (BORTHAKUR, 2015)

2.4.1 Replicação de arquivos

O HDFS é projetado para armazenar arquivos grandes em um ambiente clusterizado que pode ser escalável a milhares de DataNodes. O DataNode armazena cada arquivo como uma sequência de blocos, onde todos os blocos do arquivo, são distribuídos entre os nós. Os blocos são então replicados para garantir tolerância à falha (Fig. 5). O tamanho do bloco, bem como a quantidade de replicações que o arquivo recebe são configurações que estão presentes no NameNode.

O nó mestre é quem toma todas as decisões quanto à replicação dos blocos entre os DataNodes. Periodicamente, cada DataNode do cluster envia ao NameNode um *Heartbeat* e o relatório de blocos que estão hospedados em seus discos. Com o batimento cardíaco do servidor escravo, o NameNode tem a certeza de que este nó está funcionado corretamente. Já através do *Blockreport*, o DataNode envia a confirmação de todos os blocos pertencentes a determinados arquivos, que estão armazenados nele. (BORTHAKUR, 2015)

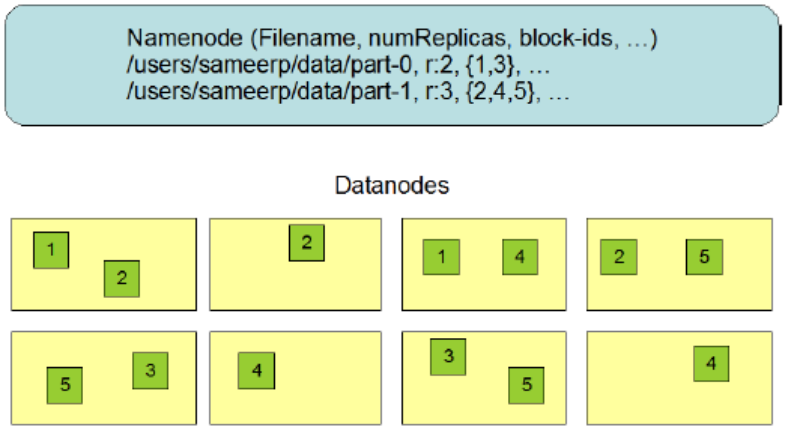

Figura 5 Replicação dos blocos.

Fonte: Borthakur, 2008.

2.4.2 Blocos de arquivo

Na arquitetura de armazenamento, existe um elemento chamado de tamanho de bloco, este representa a menor quantidade de dados que pode-se ler ou escrever em um determinado disco. Geralmente esse valor é 512 bytes nas soluções mais usuais, entretanto no HDFS esse valor passa a ser de 64 MB. Deste modo, reduz-se o custo de pesquisa no disco, também conhecido como *Seek Time*, otimizando a leitura e escrita de arquivos muito grandes.

O HDFS divide os arquivos em blocos, por padrão essa fatia tem 64 MB, todavia é possível aumentar este valor. Existem algumas aplicações que usam blocagem de 128 MB, facilitando o tratamento de grandes arquivos. (BORTHAKUR, 2015)

Outra característica importante dos blocos em HDFS, é a replicação citada anteriormente, que propicia tolerância à falha e disponibilidade dos dados em caso de alguma pane em um DataNode. Para assegurar a disponibilidade dos dados em caso de corrupção de blocos, falha em um disco ou falha em um servidor, cada bloco é replicado 3 vezes. Assim, caso seja necessário ler um bloco corrompido, a leitura pode ser feita por outro DataNode que recebeu a replicação destes dados.

2.4.3 Anatomia de leitura do HDFS

Para descrever como ocorre o processo de leitura de um arquivo no sistema de arquivos HDFS, considere a Fig. 6, que mostra a sequência de eventos que compõe esse processo.

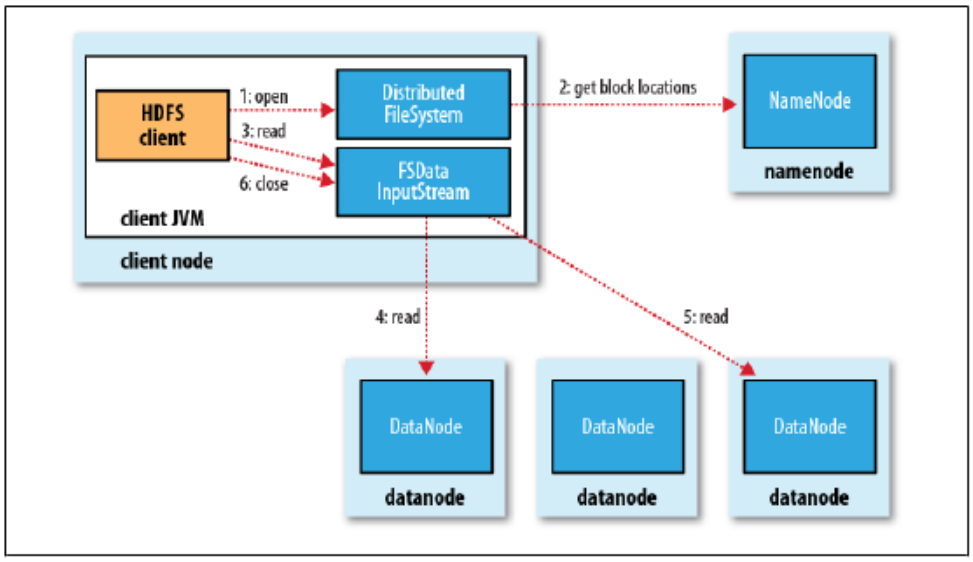

Figura 6 Processo de leitura em HDFS.

Fonte: White, 2012.

O cliente do HDFS envia a instrução de abertura para comunicar a requisição de leitura de um determinado arquivo (passo 1). O sistema de arquivos distribuído realiza uma chamada de procedimento remoto (RPC) ao NameNode que indica a localização dos blocos que compõem o arquivo (passo 2). Para cada bloco, o servidor mestre retorna o endereço de cada DataNode que possui uma cópia do bloco, além disso, os nós escravos são organizados e escolhidos a servir a requisição de acordo com a proximidade de rede com o cliente.

No passo seguinte (passo 3) o cliente fecha a comunicação direta entre o DataNode que possui o primeiro bloco do arquivo, repetindo a comunicação com os demais nós que armazenam os blocos restantes para leitura do arquivo (passos 4 e 5). Quando os blocos estão organizados e o cliente finaliza sua operação de leitura, o mesmo envia uma chamada de fechamento (passo 6). (WHITE, 2012)

2.4.4 Anatomia de escrita do HDFS

Para ilustrar como ocorre o processo de escrita de um arquivo no sistema de arquivos HDFS, avalie a Fig. 7, que particulariza a sequência de eventos que compõe esse procedimento.

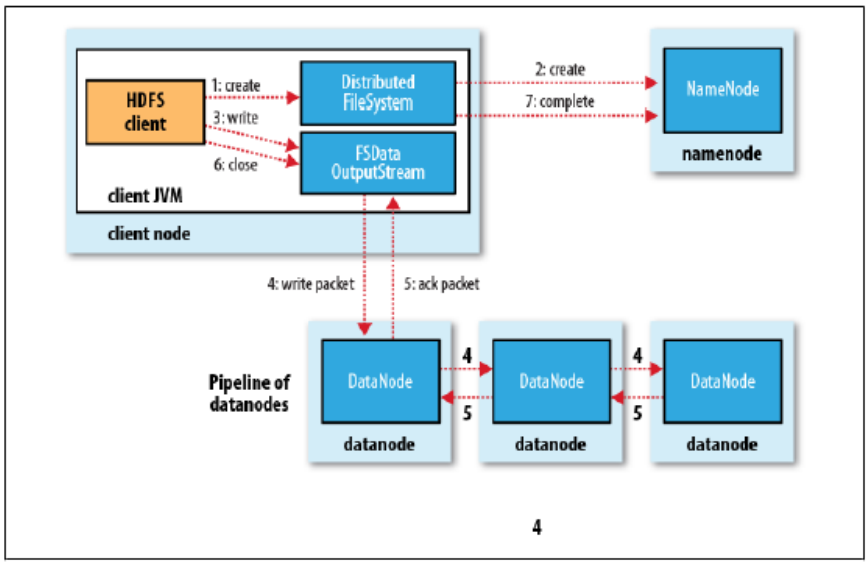

Figura 7 Processo de escrita em HDFS.

Fonte: White, 2012.

O cliente envia uma requisição de criação ao sistema de arquivos distribuído (passo 1). O sistema de arquivos distribuído realiza uma chamada de procedimento remoto (RPC) ao NameNode solicitando a criação de um novo arquivo no *namespace* (passo 2). O NameNode executa então uma varredura nos DataNode para garantir que o arquivo não existe e que não possui blocos atrelados a ele. Com a certeza de que não existe o arquivo duplicado no cluster, o cliente inicia a escrita do arquivo que é fatiado em blocos, que são distribuídos entre os DataNodes, os quais realizam em paralelo a replicação dos blocos.

No momento que ocorre a distribuição dos blocos, os registros são armazenados nos DataNodes (passo 4) e recebem entre si pacotes com o reconhecimento daquela operação (passo 5). Quando o arquivo é armazenado por completo e o cliente finaliza sua operação de escrita, o mesmo envia uma chamada de fechamento (passo 6). (WHITE, 2012)

2.5 Trabalhos correlatos

Embasado no princípio que o tempo de resposta em operações de leitura e escrita está diretamente relacionado ao tamanho do arquivo, Dong et al. (2014) traz fórmulas matemáticas que ilustram como essa relação se comporta. Ao final, são

desenvolvidos modelos de desempenho dinâmicos através das características e comportamentos de leitura e escrita presentes em ambientes HDFS, as quais foram analisadas em conjunto com outras variáveis como tamanho de bloco e largura de banda. Esse artigo embasa a proposta aqui apresentada, mostrando os detalhes da estrutura do HDFS, para que seja possível a partir de outros conhecimentos propor uma solução para o problema dos arquivos pequenos.

Li, Lin e Wang (2013) abordam sobre o impacto que o armazenamento massivo de arquivos pequenos causam ao ambiente Hadoop. De acordo com os autores, esse *framework* foi baseado em um modelo feito pelo Google para armazenar grandes arquivos, porém tem enfrentando alguns desafios, pois seu uso tem-se disseminado, principalmente, em redes sociais, nas quais o uso de arquivos pequenos como fotos e vídeos é muito comum. Plataformas que realizam grandes cargas de upload, entretanto, são arquivos de baixa volumetria. Assim utilizam a técnica denominada HMPI (Hadoop *Multimedia Processing Interface*), para processar com um bom nível de desempenho, volumes massivos de arquivos pequenos. Isso é possível, pois como ocorre com o HIPI (Hadoop *Image Processing Interface*), a arquitetura de armazenamento concatena os arquivos em um pacote (*bundle*) e indexa os meta-dados de vídeos e fotos, ao passo que as instruções de MapReduce aceleram o processamento distribuído.

Já Islam et al. (2015) propõem para melhorar a proteção dos dados, a revisão de como ocorre a replicação dos segmentos de bloco de dados, os quais são espalhados entre os nós do cluster Hadoop que, por sua vez, depende da rede de comunicação e do tipo da tecnologia do hardware de armazenamento presente para obter desempenho. Para tal apresentam o Triplo-H (o nome desta solução vem da combinação de “*Hybrid design of HDFS with Heterogeneous storage*”), que prevê o uso de tecnologia heterogêneas como RAM-Disk, SSD e HDD em clusters HPC (*High-Performance Computing*). Através deste modelo, é possível dividir em camadas, orientadas ao desempenho e considerando o grau de importância do dado ao sistema, cada tipo de arquivo.

3 SOLUÇÃO PROPOSTA

Nesse capitulo apresenta-se opções de estruturação de armazenamento de um Hadoop *Distributed File System* dado um cenário local, onde há presença massiva de arquivos pequenos (menores que 64 MB). Serão opções de boas práticas, incluindo a organização de diferentes tecnologias de persistência de dados, afim de garantir desempenho na escrita e leitura dos dados e, por consequência desempenho nas aplicações que utilizam esse sistema de arquivos distribuído.

3.1 Descrição do problema

Como citado, o HDFS foi desenvolvido para manipular arquivos grandes e, por esse motivo raiz não tem bom desempenho com arquivos considerados pequenos. Arquivos denominados pequenos são todos aqueles que possuem volumetria menor do que o bloco configurado no HDFS. Por padrão, o tamanho do bloco no HDFS é 64 MB, porém esse valor pode ser alterado para índices superiores, de acordo com a aplicação.

Existe uma grande variedade de motivos que possibilitam um cenário de enxurrada de pequenos arquivos em HDFS. O primeiro deles está na utilização deste sistema de arquivos para persistir milhares de arquivos, como fotos e pequenos vídeos que são copiados diretamente em HDFS sem modificações. Outro fator que potencializa a criação de pequenos arquivos está na utilização do HDFS em aplicações do tipo *near real-time*, onde ocorrem pequenas cargas de dados, por período.

A manipulação de arquivos pequenos atinge, principalmente, a utilização de memória do NameNode. Cada bloco no Hadoop é representando por um objeto na memória do NameNode, o qual aloca 150 bytes. Deste modo, analisando a Fig 8, caso tenha-se um arquivo menor que 1 MB, esse irá consumir da memória do NameNode a mesma quantidade que um arquivo de 64 MB, desperdiçando assim, recursos valiosos em um ambiente voltado à *Big Data.*

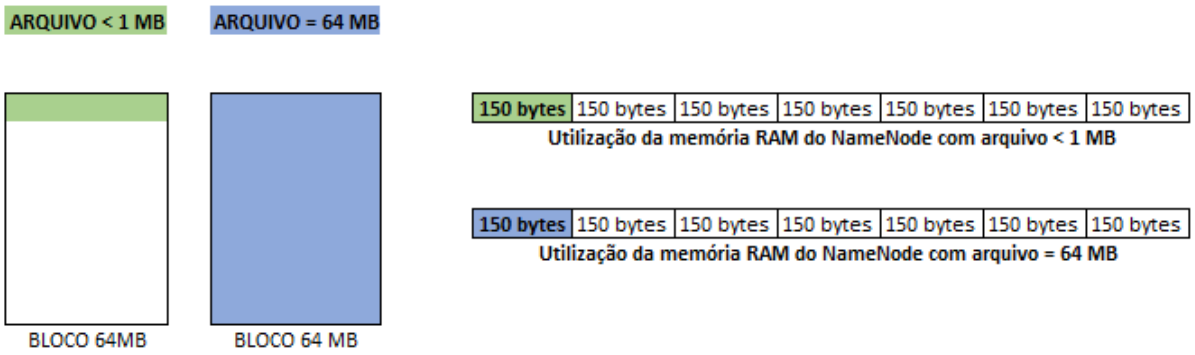

Figura 8 Consumo de memória RAM por objeto.

Fonte: Elaborada pelo autor

3.2 Tratativa do problema

O ponto inicial que será abortado nesse trabalho está ligado à tradução das necessidades das aplicações que utilizam o HDFS. Na fase de definições, o profissional responsável pelo *design* da solução de armazenamento receberá, inicialmente, a taxa esperada de vazão, nas operações de leitura e escrita do sistema. O entendimento das variáveis que compõem um cenário HDFS é de extrema importância, pois muitos fatores influenciam diretamente no desempenho deste sistema de arquivo distribuído. A variável central desta discussão está na relação entre o tamanho do arquivo e as operações de leitura e escrita.

Apoiado em um modelo dinâmico, o qual é abalizado na identificação do sistema para estabelecer comportamentos de desempenho nas operações de leitura e escrita do HDFS. Essa análise permeia diferentes cenários, para que seja validada a eficácia desta abordagem.

Pontua-se que a taxa de transferência é a taxa de vazão média realizada nas operações de leitura e escrita, TR_{rd} e TR_{wr}, respectivamente, cuja unidade computacional é medida em MB/s (*megabytes* por segundo). Representadas pelas equações 1 e 2: (DONG et al., 2014)

$$TR_{wr} = \frac{L_{wr}}{T_{wr}} \quad (1) \qquad TR_{rd} = \frac{L_{rd}}{T_{rd}} \quad (2)$$

Onde L_{wr} representa o tamanho do arquivo escrito, T_{wr} representa o tempo consumido pela operação. Já o L_{rd} corresponde ao tamanho do arquivo lido, enquanto o T_{rd} descreve o tempo consumido na instrução de leitura.

Sendo K o tamanho do arquivo, a equação 3 apresenta o tempo gasto para uma operação de escrita no HDFS.

$$T_{wr}(k) = T_{cre} + \left\lceil \frac{k}{BS} \right\rceil * \left(T_{adb} + T_{red} + T_{pip}\right) + \sum_{i=1}^{\lceil k/PS \rceil} T_{pac_i} + T_{cpl} \quad (3)$$

Onde T_{cre} é o tempo gasto pela criação do meta-dado no sistema de arquivos namespace, presente no NameNode. T_{adb} representa o tempo gasto para replicação dos blocos, que por padrão divide-se em três DataNodes. O tempo gasto para que o NameNode receba a lista dos DataNodes utilizados é denominado T_{red}. O período gasto pelo processo de comunicação do tipo socket do cliente HDFS com os DataNodes recebe o valor T_{pip}. O tempo necessário pela fragmentação dos blocos, bem como a persistência dos dados, aviso de conclusão de operação e verificação via *checksum* recebe o valor T_{pac_i}. Com o término da instrução de escrita, o tempo necessário para finalizar a conexão e verificar as réplicas é chamado de T_{cpl}. (DONG et al., 2014)

Deste modo, a taxa de transferência quando escreve-se um arquivo de tamanho k é dado pela equação 4:

$$TR_{wr}(k) = \frac{k}{T_{cre} + \left\lceil \frac{k}{BS} \right\rceil * \left(T_{adb} + T_{red} + T_{pip}\right) + \sum_{i=1}^{\lceil k/PS \rceil} T_{pac_i} + T_{cpl}} \quad (4)$$

Onde *BS* é o tamanho do bloco e *PS* é o tamanho do pacote.

De forma semelhante à escrita, a equação 5 representa o tempo para leitura de um arquivo de tamanho K.

$$T_{rd}(k) = \left\lceil \frac{k}{BS * pre} \right\rceil * \left(T_{asl} + T_{rel}\right) + \left\lceil \frac{k}{BS} \right\rceil * T_{asb} + \sum_{i=1}^{\lceil k/BS \rceil} T_{reb_i} \quad (5)$$

Onde T_{asl} é o tempo gasto pela requisição do cliente, localização dos blocos repartidos pelos DataNodes, os quais são ordenados pela distância na topologia de rede. T_{rel} representa o período gasto pelo NameNode para retornar a requisição de

localização dos blocos. A cada bloco, o cliente envia uma requisição de leitura para o DataNode com menor distância de rede, chamado de T_{asb}. T_{reb} é o tempo gasto para busca dos pacotes até a conclusão da transferência do bloco para o cliente. Por último, *pre* representa a quantidade de blocos que cada instrução trata paralelamente, por padrão, no HDFS são tratados 10 blocos.

A taxa de transferência para a operação de leitura de um arquivo de tamanho *k* é dado pela equação 6: (DONG et al., 2014)

$$TR_{rd}(k) = \frac{k}{\left\lceil \frac{k}{BS*pre} \right\rceil * (T_{asl} + T_{rel}) + \left\lceil \frac{k}{BS} \right\rceil * T_{asb} + \sum_{i=1}^{\lceil k/BS \rceil} T_{reb_i}} \quad (6)$$

Uma vez que foram identificadas as variáveis presentes no ambiente HDFS, os passos seguintes têm como principal objetivo, a redução do tempo gasto, consequentemente, melhores taxas de vazão são alcançadas.

Deste modo, o segundo ponto que será tratado está relacionado ao consumo de processamento e memória que arquivos de vídeo e fotos realizam no *cluster* Hadoop, em especial no HDFS. Os processos que o HDFS realiza em instruções de leitura e escrita seguem o roteiro descrito no Capítulo 2, onde o NameNode é responsável por orquestrar todas as ações, as quais são armazenadas em memória e sofrem penalidade quando se trata de arquivos pequenos.

Para exemplificar o consumo de memória de arquivos pequenos em HDFS, foram utilizados 10 milhões de arquivos do tipo imagem, armazenados como objetos, com tamanho variando entre 1 KB e 200 KB. Esse acervo consumirá 2 GB de memória de um NameNode, valor esse considerado alto tendo em vista que a mesma quantidade de memória poderia processar uma volumetria muito superior, porém em arquivos grandes.

Pode-se utilizar do modelo de processamento HMPI (Hadoop *Multimedia Processing Interface*) para melhorar o desempenho do HDFS para armazenar arquivos de multimídia pequenos com eficiência. É através de uma interface única entre o cliente e o cluster Hadoop que as imagens e vídeos são classificados automaticamente. A seguir, esses arquivos pequenos se fundem e ocorre a criação de um novo arquivo,

no entanto, um arquivo grande com o tamanho do bloco configurado. Nesse cenário, as mídias possuem seus meta-dados armazenados em um arquivo do tipo índice, atrelando a vantagem de processamento paralelo com uso do MapReduce, tornando assim, melhor a experiência no acesso aos arquivos de imagens e vídeos. (LI; LIN; WANG, 2013)

Nota-se na Fig. 9 que modelo HMPI é composto por três principais camadas: uma interface usuária responsável por unificar as requisições dos usuários comuns, atendendo as solicitações para carregar, baixar arquivos ou procurar arquivos que estejam persistidos no repositório HDFS (passo 1); uma camada de processamento do modelo HMPI a qual possui as funções de identificar o tipo e tamanho de arquivos multimídia, distribuir a segmentação de vídeo e fundir os arquivos (passo 2), bem como interagir com o cliente HDFS (passo 3); por último, a camada do HDFS persiste os arquivos gerados, frutos do HMPI, de forma eficiente (passo 4).

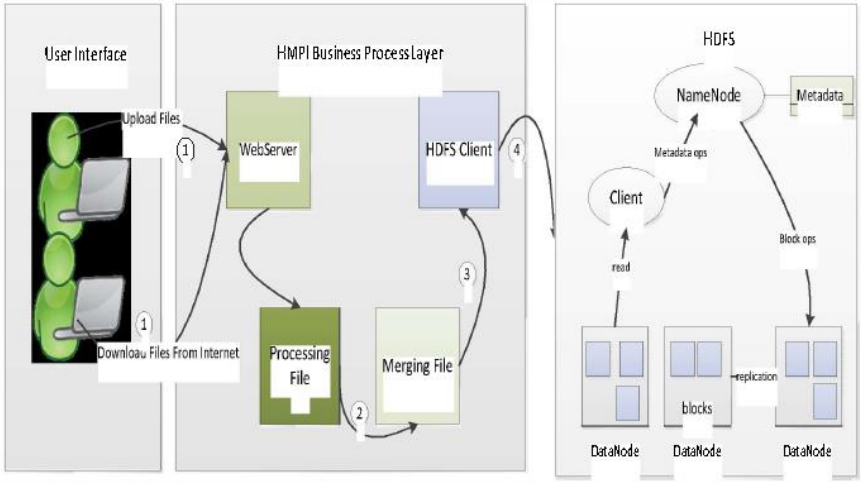

Figura 9 Arquitetura do modelo HMPI em 3 camadas.
Fonte: Li, Lin e Wang, 2013.

A ideia principal para projetar o HMPI foi de facilitar a experiência do HDFS na utilização deste sistema de arquivos distribuído para manipular e armazenar arquivos de imagens e vídeos, ao passo que a quantidade de memória necessária é reduzida, pode-se processar mais informações a custos reduzidos de *hardware*.

Outra proposta para melhorar o desempenho no uso de grande quantidade de arquivos pequenos aqui apresentada consiste no uso de tecnologias heterogêneas de armazenamento para minimizar os gargalos que o HDFS possui em sua forma de arquitetura padrão. Utiliza-se como base, os estudos realizados por Islam et al. (2015) que promovem um modelo híbrido de persistência dos dados, através de discos do tipo RAM-Disk, SSD e HDD, orientado ao desempenho e relevância.

No cenário híbrido proposto, observa-se que as tecnologias de armazenamento possuem características distintas e o grande diferencial entre elas está na largura de banda máxima, ou em outras palavras, o quanto de vazão ela é capaz de garantir. Na Fig. 10, é possível observar os valores que diferenciam esses hardwares, entre desempenho e capacidade.

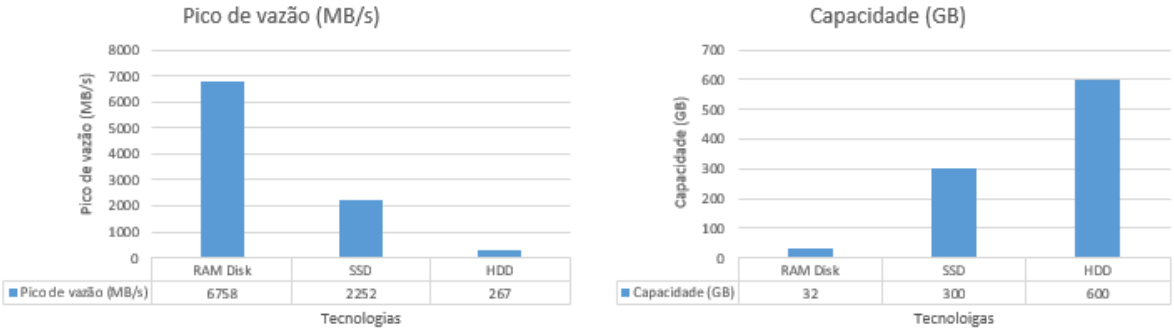

Figura 10 Comparativo entre tecnologias.

Fonte: Adaptado de Islam et al., 2015

Por meio do modelo de arquitetura de armazenamento chamado de Triplo-H, os gargalos de I/O são minimizados por ações conhecidas como *buffering* e *caching*. Em resumo, essas ações preveem a utilização de memória RAM ou dispositivos de armazenamento que possuam desempenho próximo à memória RAM, para persistir os dados mais recentes ou aqueles que são mais acessados pelo ambiente.

A hierarquia de armazenamento, relacionada na Fig. 10 presente no modelo Triplo-H, têm três principais tipos de *hardware*:

- **RAM-Disk** que é um dispositivo baseado em memória, mais rápido que os discos SSDs em pelo menos 3x que suportam altas cargas de leitura e escrita.

- **Discos de estado sólido**, chamados de SSD, proporcionam armazenamento com desempenho elevado e são essencialmente adequados à aplicações em *Big Data*.
- **Unidades de disco rígido**, chamado de HDD, aprovisionam a maior quantidade de armazenamento de dados, entretanto são lentos em termos de desempenho em comparação com as outras duas tecnologias citadas.

A proposta consiste em se empregar os dispositivos do tipo RAM-Disk como primeira camada de buffer-cache, além de aproveitar os SSDs para aumentar a primeira camada, eles serão importantes para assegurar os dados em um cenário de falha, visto que os discos do tipo RAM-Disk possuem sensibilidade à falta de fornecimento elétrico. Por último, serão empregados os discos do tipo HDD, pois armazenar toda a volumetria de um ambiente *Big Data* em discos rápidos seria muito caro e, impossibilitaria o vasto uso desta solução. (ISLAM et al., 2015)

Os principais artefatos desta arquitetura híbrida de persistência de dados em HDFS são:

- Um seletor de política junto ao cliente HDFS que atribui peso a diferentes arquivos, conforme configuração e necessidade do negócio/aplicação.
- Políticas de posicionamento de dados que determinam as regras para utilizar de forma otimizada os diferentes tipos de armazenamento disponíveis no *cluster*.
- Um mecanismo de posicionamento de dados escolhe o tipo do dispositivo de armazenamento apropriado, baseado nos pesos gerados pelo cliente HDFS, além de avaliar a disponibilidade da capacidade de armazenamento necessária, grava os dados no dispositivo determinado. É também esse artefato responsável por detectar dados que não são críticos e, por consequência, não necessitam de desempenho e devem ser persistidos em discos do tipo HDD.

No processo que envolve as políticas de posicionamento, a arquitetura proposta é subdivida em dois cenários, os quais são ilustrados na Fig. 11. A primeira forma é chamada de Posicionamento Ganancioso (*Greedy Placement*) que consiste em

gravar todas as entradas na camada de armazenamento mais nobre, deste modo, enquanto houver espaço no RAM-Disk ele é utilizado, passando a persistir o restante dos dados nas outras tecnologias disponíveis. Em um cenário que exemplifica essa forma, dá-se um arquivo F_i que foi segmentado em dois blocos B_{i1} e B_{i2}, onde i é a identificação do arquivo. Desta maneira, os blocos pertencentes aos arquivos hipotéticos F_1, F_2, F_3 e F_n irão ocupar, inicialmente, os discos mais rápidos. Esse método de posicionamento é indicado para ambientes onde o grau de importância do arquivo respeita, entre outras coisas, especialmente, a linha do tempo.

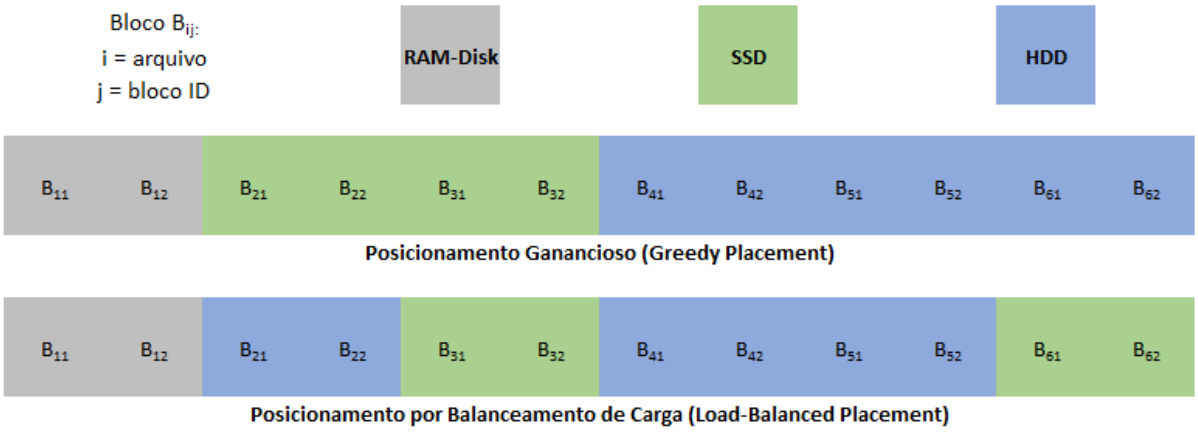

Figura 11 Politicas de posicionamento.

Fonte: Adaptado de Islam et al., 2015

Já na forma, conhecida como Posicionamento por Balanceamento de Carga (*Load-balanced Placement*) todas as instruções são pulverizadas entre os dispositivos elencados através da classificação de peso do arquivo, que visa garantir que os arquivos mais sensíveis ao desempenho estarão na tecnologia mais nobre. Deste modo, os arquivos F_1, F_2, F_3 e F_N terão seus blocos persistidos em dispositivos, respeitando a classificação do peso dado a eles. Além de ser a forma mais indicada para grande parte dos cenários, onde o HDFS é utilizado, é também a maneira que traz o melhor aproveitamento das tecnologias presentes. No modelo Ganancioso, todos os arquivos são armazenados, inicialmente, em dispositivos nobres e, posteriormente, são rebaixados quando avaliada sua importância. Contudo o

Posicionamento por Balanceamento de Carga otimiza essa classificação, evitando problemas como armazenamento de réplicas em discos do tipo RAM-Disk.

3.3 Resultados obtidos

Os resultados obtidos empregando as propostas de concatenação dos arquivos pequenos pelo modelo HMPI e a de utilização de tecnologias heterógenas para persistências dos dados em HDFS podem ser observados a seguir.

Os testes do modelo HMPI foram realizados por Li, Lin e Wang (2013) construindo um *cluster* Hadoop utilizando-se de 5 computadores. Um nó atuando como NameNode com um processador Intel Xeon E5606 CPU 2.13GHz, 8GB memória e 1 TB disco HDD. Enquanto os DataNodes possuíam um processador Intel Core i3 CPU 2.93GHz, 4GB memória e 500GB de disco HDD. Em cada nó, foi instalado o sistema operacional CentOS Server 6.3 com *kernel* 2.6.32-279, Hadoop 1.0.3, Java 1.6.25, com três replicações e bloco HDFS com tamanho de 64 MB.

Tendo ciência de que o número grande de arquivos de mídia afeta o desempenho do HDFS, em especial a alocação de memória RAM, foram utilizados conjuntos de arquivos que possuem 3000, 6000, 9000, 120000, 15000, 18000 e 21000 imagens. Para o experimento com vídeos, foram utilizados 100 vídeos com 2 GB cada, divididos em segmentos de 210 MB, compondo conjuntos de 200, 400, 600,

800 e 1000 segmentos.

Os resultados da administração de arquivos de imagens em relação ao HDFS padrão podem ser observados nas Figs. 12 e 13 a seguir:

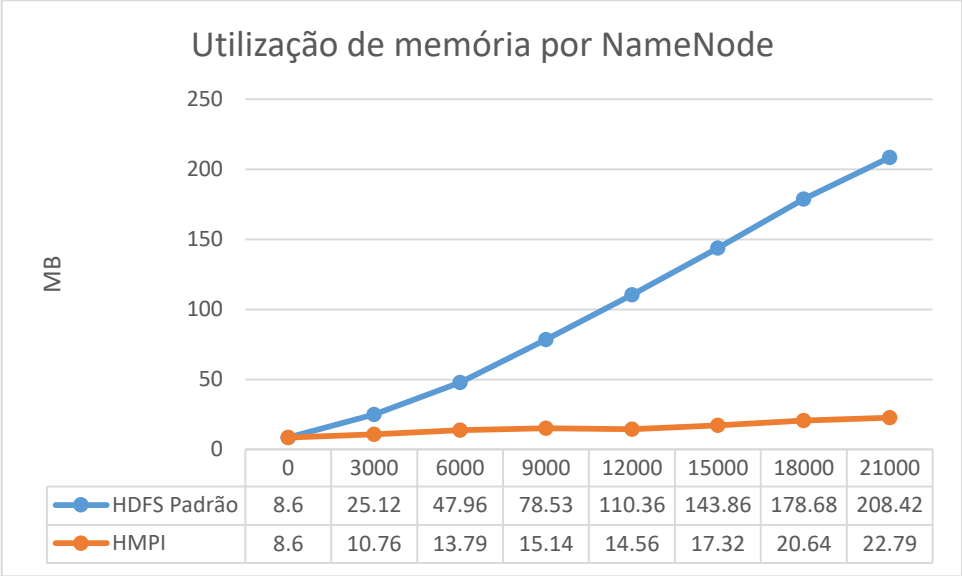

Figura 12 Uso de memória do NameNode.

Fonte: Adaptado de Li, Lin e Wang, 2013.

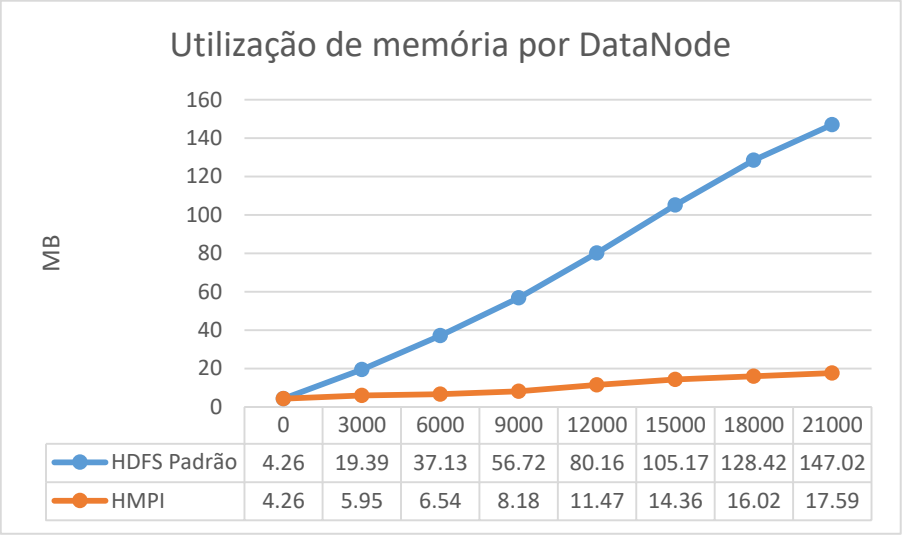

Figura 13 Uso de memória do DataNode.

Fonte: Adaptado de Li, Lin e Wang, 2013.

Já em relação aos arquivos de vídeo, não houve significativa melhora do armazenamento de arquivos de vídeo, bem como o uso de memória de NameNode, visto que os arquivos de vídeo geralmente possuem tamanhos superiores ao tamanho do bloco padrão, que é de 64 MB. Porém, o uso do HMPI fornece uma interface única para armazenar e acessar arquivos de vídeo, o qual garante a vantagem de utilizar processos MapReduce do Hadoop para ler e processar

arquivos vídeo, provendo diminuição no tempo de resposta, como é possível ver a seguir na Fig. 14**.**

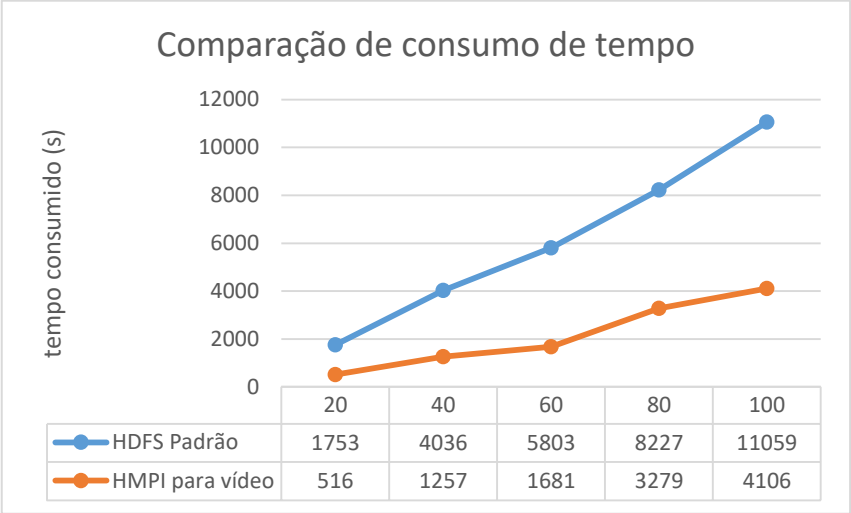

Figura 14 Comparação de consumo de tempo.

Fonte: Adaptado de Li, Lin e Wang, 2013.

Conclui-se por meio das experiências de Li, Lin e Wang (2013) que o modelo HMPI promove a diminuição da utilização de memória do NameNode e dos DataNodes, ao passo que os arquivos pequenos são tratados como arquivos maiores, por meio de um processo de fusão. O experimento ainda mostra como é possível otimizar a utilização do HDFS para administrar a enxurrada de arquivos de mídia oriundos de redes sociais. Esse exemplo, coloca essa forma de organização à frente das demais soluções de armazenamento.

Nos experimentos relacionados às tecnologias heterogêneas de armazenamento, Islam et al. (2015) utilizaram três *clusters* distintos para comprovar a eficácia do modelo Triplo-H em relação ao HDFS padrão:

- ***Cluster* A** – O Intel Westmere Cluster é composto por 9 nós, cada nó possui processador Xeon Dual 4-core de 2.67 GHz, 24 GB de RAM, dois discos HDD de 1 TB, um disco SSD de 300 GB além de 12 GB em RAM Disk. O sistema operacional é o Red Hat Enterprise Linux Server 6.1.
- ***Cluster* B** – O SDSC Gordon é composto por 1024 nós de processamento e 64 I/O nós, cada *compute node* possui dois processadores 8-core 2.6

GHz Intel EM64T Xeon E5, 64 GB de RAM, 16 discos SSD de 300 GB e 32 GB em RAM Disk. O sistema operacional é o CentOS 6.4.

- ***Cluster* C** – Cada nó do TACC Stampede possui 2 processadores 8-core Intel Sandy Bridge de 2,70 GHz, 32 GB de RAM, um disco HDD de 80 GB e outro RAM Disk de 16 GB. Foi escolhido o CentOS 6.3 como sistema operacional.

Em todos os *clusters* a versão do Hadoop instalado foi a versão 2.6.0 com JDK 1.7.0. Nos testes realizados com os *Clusters* A e B foram utilizadas as tecnologias de armazenamento RAM Disk, SSD e HDD. Entretanto, o *Cluster* C não possui discos do tipo SSD e, por esse motivo não há uma camada intermediaria para a persistência dos blocos oriundos do HDFS. Essa ausência é válida para o estudo aqui apresentado, visto que pode-se encontrar esse tipo de limitação no dia-a-dia.

O primeiro experimento realizado foi através, do *benchmark* TestDFSIO, utilizado largamente em teste de estresse do HDFS em operações de leitura e escrita. A Fig. 15 mostra os valores obtidos por meio do *Cluster* C, onde foram empregados 40 GB de dados em 8 DataNodes, 80 GB em 16 DataNodes e 160 GB em 32 DataNodes. Ainda na Fig. 15, pode-se comprovar que o modelo Triplo-H obteve desempenho 7 vezes maior do que o HDFS padrão nas operações de escrita. Superioridade essa garantida através da larga camada de *buffer-cache.*

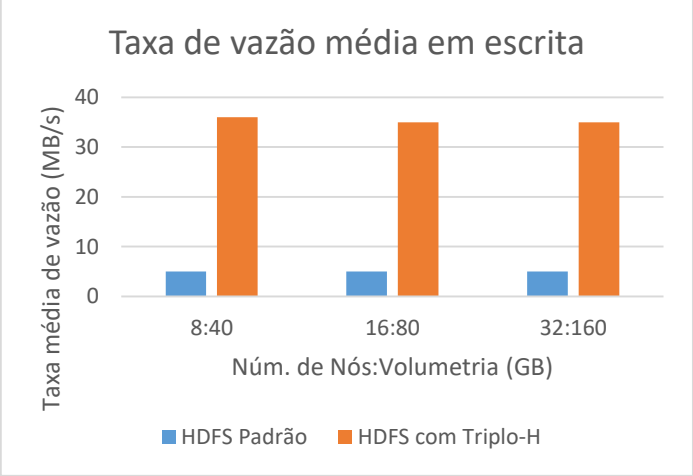

Figura 15 Avaliação do TestDFSIO no Cluster C em escrita.

Fonte: Adaptado de Islam et al., 2015.

Nas operações de leitura, o Triplo-H alcançou marcas 2 vezes melhores do que o HDFS com configuração nativa, graças a redução do número de I/O. Os valores obtidos estão apresentados na Fig. 16.

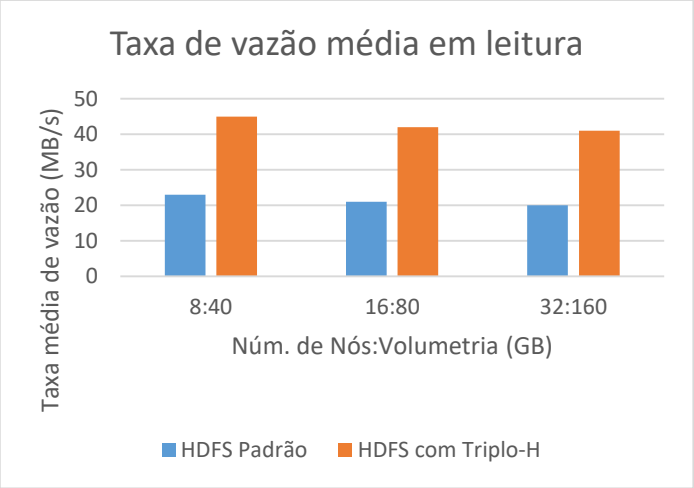

Figura 16 Avaliação do TestDFSIO no Cluster C em leitura.

Fonte: Adaptado de Islam et al., 2015.

Foram utilizados outras aplicações de *benchmark*, além do TestDFSIO para testar o comportamento do Triplo-H em relação ao HDFS. Para criação de arquivos, foram aplicados as ferramentas: TeraGen, RandomTextWriter, e RandomWriter.

O RandomTextWriter foi aplicado em 8 nós do *Cluster* A, onde o Triplo-H obteve redução no tempo de execução em 48% no volume de 60 GB em relação ao HDFS sem tecnologia heterogênea, conforme Fig. 17:

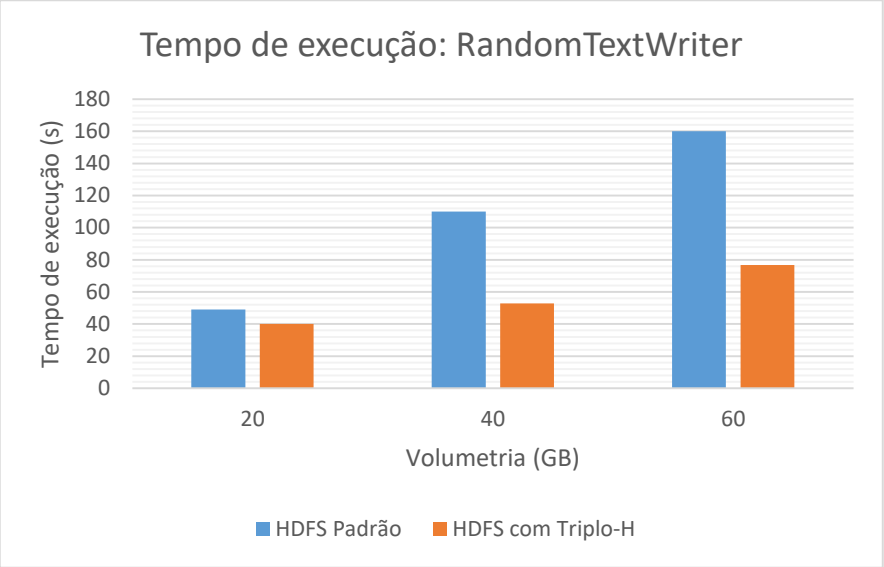

Figura 17 Tempo de execução do RandomTextWriter.

Fonte: Adaptado de Islam et al., 2015.

Já através da aplicação do TeraGen, em 32 nós do *Cluster* B, o desempenho do ambiente Triplo-H foi superior em 42%, comparando com outro ambiente sem essa proposta, de acordo com a Fig. 18.

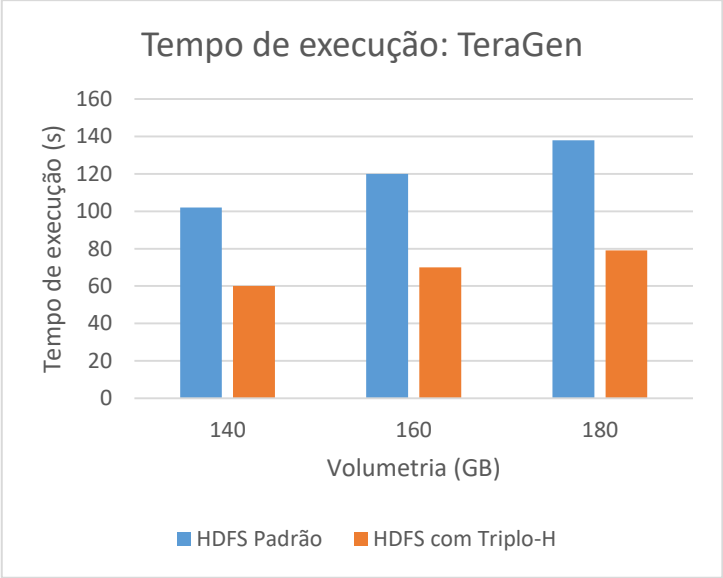

Figura 18 Tempo de execução do TeraGen.

Fonte: Adaptado de Islam et al., 2015.

Na Fig. 19, 32 nós do *Cluster* C foram utilizados nos testes com RandomWriter, onde pode-se observar que houve redução do tempo de execução em 3 vezes, com o Triplo-H.

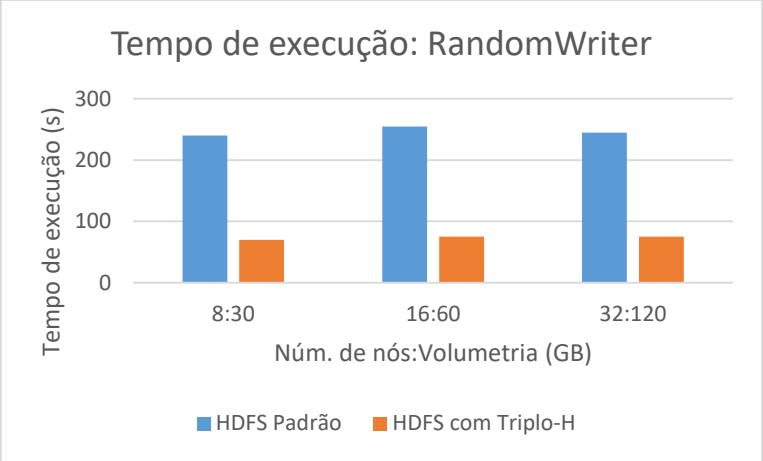

Figura 19 Tempo de execução do RandomWriter.

Fonte: Adaptado de Islam et al., 2015.

Por último, foi evidenciada a diferença de comportamento entre o modelo de Posicionamento Ganancioso (*Greedy Placement*) e o Posicionamento por Balanceamento de Carga (Load-balanced Placement). No comparativo, foram realizadas operações de escrita de um volume de 30 GB em 4 nós dos *Clusters* A e C, com 2, 4 e 8 execuções de MapReduce simultâneas. Conforme Fig. 20, no *Cluster* A, o modelo Ganancioso desempenhou melhor que o modelo por Balanceamento de Carga em 2 execuções concorrentes. A partir de 4 execuções, o modelo que balanceia a carga obteve melhores resultados. Já o experimento com o *Cluster* C (que não possui discos SSD), demonstrou que o modelo Ganancioso trouxe taxas de vazão superiores em todas as execuções simultâneas, uma vez que nesse cenário não há a presença de tecnologia intermediaria para persistência dos blocos (Fig. 21).

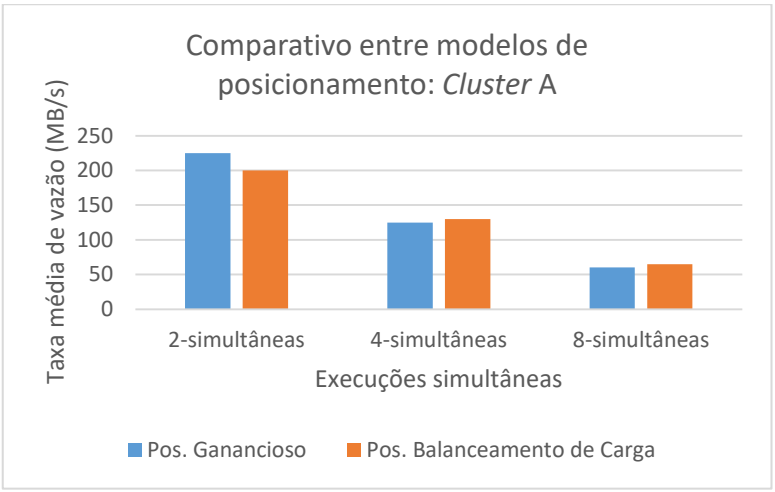

Figura 20 Resultados do TestDFSIO no *Cluster* A em escrita.

Fonte: Adaptado de Islam et al., 2015.

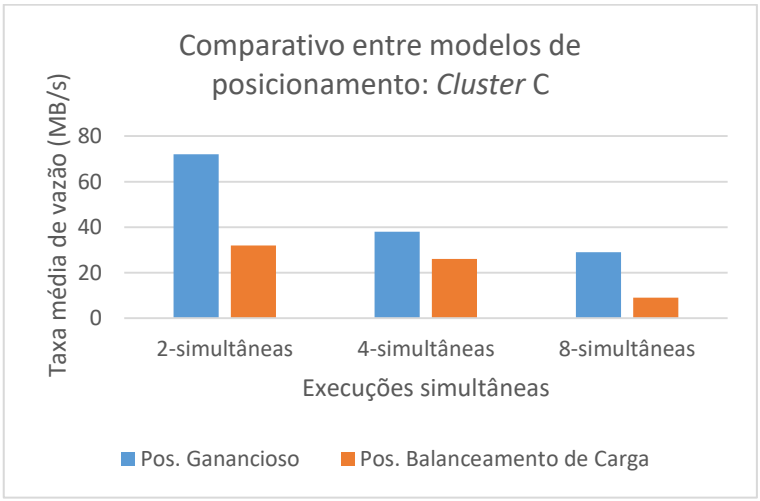

Figura 21 Resultados do TestDFSIO no Cluster C em escrita.

Fonte: Adaptado de Islam et al., 2015.

4 CONCLUSÃO

A definição de uma arquitetura de armazenamento que garanta bom desempenho, é premissa para o ciclo de vida de qualquer solução de armazenamento. As dificuldades encontradas no ambiente selecionado, evidenciam que apesar de um sistema de arquivos que foi desenvolvido para *Big Data*, é de extrema importância uma análise profunda e contínua do desempenho deste em diferentes formas de utilização, garantindo assim, sobrevida e amplo emprego.

Inicialmente, o HDFS foi desenvolvido para manipulação de grande quantidade de arquivos grandes, ou seja, que superem o tamanho da blocagem padrão. Entretanto, a utilização deste sistema de arquivos distribuídos tem se expandindo para o armazenamento de dados advindos de redes sociais e aplicações do tipo near real-time, como a Internet das Coisas (IoT), que geram enxurradas de arquivos pequenos. A experiência do HDFS padrão, nesses cenários, não otimiza a utilização da memória RAM do *cluster* Hadoop, que é sublocada.

Pode-se reforçar estas afirmações, por meio de equações matemáticas, que despontam que as taxas de transferência em um ambiente HDFS sofrem influência, principalmente, pelo tamanho do arquivo, tamanho do bloco, velocidade do *hardware* de armazenamento e desempenho da camada de rede de comunicação entre o NameNode e os DataNodes.

Os estudos e resultados presentes neste trabalho, feitas a partir de trabalhos prévios, de concatenação dos arquivos pequenos pelo modelo HMPI e a utilização de tecnologias heterógenas para persistências dos dados, trazem ao HDFS maior agilidade nas operações de escrita e leitura.

Para confirmar os benefícios das técnicas apresentadas, como trabalhos futuros, anseia-se que as duas técnicas sejam empregas em paralelo em experimentos. Isto permitiria observações, que poderiam enriquecer as equações matemáticas, visto que podem haver outros parâmetros que não foram abordados nos trabalhos prévios. O experimento não foi possível de ser realizado durante o desenvolvimento dessa monografia, pois criar um modelo que exponha todas as variáveis em um

cenário computacional, caracteriza uma ação complexa e exige um tempo maior do que o disponível para o desenvolvimento do trabalho aqui apresentado.

5 REFERÊNCIAS BIBLIOGRÁFICAS

AGARWAL, Arpit; RADIA, Sanjay; SRINIVAS, Suresh. **Heterogeneous Storage for HDFS**. Forest Hill: The Apache Software Foundation, 2013. 11 p.

BORTHAKURR, Dhruba. **HDFS Architecture Guide**. Forest Hill: The Apache Software Foundation, 2008. 13 p.

DONG, Bo et al. **Performance models and dynamic characteristics analysis for HDFS write and read operations: A systematic view**. The Journal Of Systems And Software**.** Xi'an, p. 132-151. 19 fev. 2014.

GNANASUNDARAM, Somasundaram; SHRIVASTAVA, Alok (Ed.). **Information Storage and Managemen**t: Storing Managing and Protecting Digital Information in Classic, Virtualized, and Cloud Environments. 2. ed. Indianapolis: John Wiley & Sons, Inc, 2012. 530 p.

ISLAM, Nusrat Sharmin et al. **Triple-H: A Hybrid Approach to Accelerate HDFS on HPC Clusters with Heterogeneous Storage Architecture**. Ohio: IEEE, 2015.

KLUG, Anthony; TSICHRITZIS, Dionysios C. **The ANSI/X3/SPARC DBMS Framework:** Report of the Study Group on Database Management Systems. 2. ed. New York: Computer Systems Research Group, 1975. 140 p.

LI, Jia; LIN, Kunhui; WANG, Jingjin. **Design of the Mass Multimedia Files Storage Architecture Based on Hadoop.**Colombo: The 8th International Conference On Computer Science & Education, 2013.

MICHAELIS (Brasil). **Michaelis Dicionário Brasileiro da Língua Portuguesa.** São Paulo: Editora Melhoramentos Ltda., 2016.

PRESTON, W. Curtis. **Using SANs and NAS.** Sebastopol: O'reilly & Associates, Inc, 2002. 66 p.

SNIA (Colorado Springs). **Storage Networking Industry Association.** 2016. Disponível em: <www.snia.org/>. Acesso em: 22 nov. 2016.

TAURION, Cezar. **Big Data**. Rio de Janeiro: Brasport Livros e Multimídia Ltda., 2013. 102 p.

VENNER, Jason. **Pro Hadoop**: Build scalable, distributed applications in the cloud. New York: Apress, 2009. 442 p.

WHITE, Tom. **Hadoop: The Definitive Guide**. 3. ed. Sebastopol: O'reilly Media Inc., 2012. 657 p.

XUAN, Pengfei et al. Accelerating big data analytics on HPC clusters using two-level storage. **Parallel Computing,** [s.l.], p.1-17, ago. 2016. Elsevier BV. http://dx.doi.org/10.1016/j.parco.2016.08.001.