Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Políticas de Replicação de Dados no Ambiente de Computação em Nuvem

Gabriel Heleno Gonçalves da Silva

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof.ª Dr.ª Maristela Terto de Holanda

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenadora: Prof.ª Dr.ª Maristela Terto de Holanda

Banca examinadora composta por:

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora) — CIC/UnB
Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo — CIC/UnB
Prof.ª Dr.ª Maria Emília Machado Telles Walter — CIC/UnB

CIP — Catalogação Internacional na Publicação

da Silva, Gabriel Heleno Gonçalves.

Políticas de Replicação de Dados no Ambiente de Computação em Nuvem / Gabriel Heleno Gonçalves da Silva. Brasília : UnB, 2013.
117 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2013.

1. Computação em Nuvem, 2. Políticas de Replicação,
3. Bioinformática, 4. Hadoop, 5. NoSQL

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Políticas de Replicação de Dados no Ambiente de Computação em Nuvem

Gabriel Heleno Gonçalves da Silva

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof.ª Dr.ª Maristela Terto de Holanda (Orientadora)
CIC/UnB

Prof.ª Dr.ª Aletéia Patrícia Favacho de Araújo
CIC/UnB

Prof.ª Dr.ª Maria Emília Machado Telles Walter
CIC/UnB

Prof.ª Dr.ª Maristela Terto de Holanda
Coordenadora do Bacharelado em Ciência da Computação

Brasília, 13 de Agosto de 2013

Dedicatória

Este trabalho é dedicado aos familiares, amigos e a todas as pessoas que se sacrificaram e dedicaram as suas vidas para a elaboração das tecnologias que foram utilizadas no desenvolvimento deste trabalho, mesmo que indiretamente.

Agradecimentos

O autor gostaria primeiramente de agradecer a Deus por ter a oportunidade de participar deste trabalho e, assim, poder contribuir de alguma forma com o aprimoramento da ciência. Também há de se agradecer à orientadora Maristela Terto de Holanda, à família (pai, mãe, irmã e entes queridos), amigos e a todos que ajudaram e incentivaram o desenvolvimento deste trabalho de alguma forma. Aproveita-se ainda essa oportunidade para agradecer de maneira especial aos professores da Universidade de Brasília, do Colégio Militar de Brasília e de tantas outras instituições de ensino, que compilaram na forma de pensar e agir, princípios e conhecimentos de suma importância para a realização de um trabalho de sucesso. A todos os citados, que contribuiram para a formação acadêmica de tal maneira, que puderam tornar o sonho da conclusão deste trabalho uma realidade, a minha mais sincera gratidão.

Resumo

Atualmente, muitas arquiteturas de gerenciadores de bancos de dados NoSQL (*Not only SQL*) estão no cenário computacional, cada um com sua forma particular de armazenar, localizar e alterar os dados replicados. Este trabalho propõe analisar os processos e políticas de replicação dos dados de sistemas de arquivos distribuídos a fim de propor uma política de replicação de dados própria para uma nuvem computacional de dados de bioinformática, de modo a alcançar níveis mais altos de eficiência e qualidade nos serviços prestados pela nuvem.

Palavras-chave: Computação em Nuvem, Políticas de Replicação, Bioinformática, Hadoop, NoSQL

Abstract

Currently, many NoSQL (*Not only SQL*) architectures data base managers are in the computational scenario, each with its own way of storing, finding and changing the replicated data. The present study aims to analyze the process and policies of replicating data of distributed file systems to propose its own data replication policy to a computational cloud treating bioinformatics data, in order to achieve higher levels of efficiency and quality in services provided by the cloud.

Keywords: Cloud Computing, Replication Policies, Bioinformatics, Hadoop, NoSQL

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Inúmeras tecnologias foram criadas para permitir que as pessoas pudessem trocar informações e recursos entre si de onde quer que estivessem. Exemplos dessas necessidade de compartilhamento e comunicação entre as pessoas vão desde os sinais de fumaça, a criação dos correios, das redes de telefone, e até a Internet. Dessa necessidade surgiu também a demanda por serviços que pudessem integrar tecnologias já criadas, de forma a maximizar o desempenho alcançado, e também permitissem uma expansão das redes existentes.

A medida em que o volume de dados computacionais crescem, cada vez mais serviços de persistência de dados surgem para atender diferentes demandas de seus clientes. Tais serviços dispõem de arquiteturas particulares que conferem a cada um deles diferentes modelos e políticas de replicação.

Em relação aos modelos, o mais utilizado comercialmente na atualidade é o relacional. Fazendo jus ao nome, o modelo relacional liga os seus dados por meio dos relacionamentos entre suas tabelas. Um banco de dados relacional segue o modelo ACID (atomicidade, consistência, isolamento e durabilidade), e para garantir o máximo de consistência utiliza as normalizações das tabelas, que não podem ser particionadas.

O balanceamento entre disponibilidade, consistência e tolerância ao particionamento dá origem à criação do teorema CAP (*Consistency, Availability and Partition Tolerance*) [11] que afirma que entre as características citadas acima só é possível escolher duas, ou seja, é possível formar os pares: disponibilidade e consistência, disponibilidade e tolerância ao particionamento e consistência e tolerância ao particionamento. O modelo relacional, como visto, representa o par formado pelas duas primeiras letras do teorema CAP (consistência e disponibilidade), mas ainda ficam faltando os outros dois pares que não podem ser desenvolvidos utilizando o modelo relacional.

Nesse cenário surgem como alternativa para os bancos de dados relacionais os modelos NoSQL (*Not only Structured Query Language*), que são modelos os quais não se concentram apenas em uma linguagem de consulta estruturada, mas que possibilitam que os dados sejam divididos, e em troca da tolerância ao particionamento é necessário escolher entre consistência e disponibilidade de seus dados, ou seja, na proporção que a necessidade de consistência aumenta a disponibilidade de acesso ao dado fica mais restrita.

Os bancos NoSQLs utilizam um sistema de arquivos distribuído para realizar o particionamento dos dados, e as políticas de replicação que cada um desses sistemas de

arquivos utiliza influi diretamente no desempenho, na disponibilidade e na consistência de seus dados.

A *Cloud Computing* ou Computação em Nuvem é o método de compartilhamento de recursos entre computadores ou redes de computadores, no qual o acesso a esses recursos pode ser feito de maneira remota, ou seja, na nuvem [27]. Nunca uma abordagem para a utilização real foi tão global e completa: não apenas recursos de computação e armazenamento são entregues sob demanda, mas também todo recurso de software e transmissão pode ser aproveitado na nuvem [31]. Tendências anteriores à computação em nuvem foram limitadas a uma determinada classe de usuários ou focadas em tornar disponível uma demanda específica de recursos de TI, principalmente de informática [12].

Dentre as muitas propriedades observadas na computação em nuvem, uma das questões fundamentais é a redução dos custos de atividades ligadas ao processamento e armazenamento de dados, visto que recursos de grandes *data centers* podem ser utilizados para processos fora de seus domínios, evitando o desperdício dos mesmos e ainda possibilitando que as empresas façam a venda do excedente de suas capacidades operacionais. Para muitas empresas, especialmente para *start-ups* e médias empresas, o pagamento baseado no uso do modelo de computação em nuvem, juntamente com o suporte para manutenção do hardware é muito atraente [2], sendo um dos motivos que tem possibilitado a pequenos empreendimentos se expandir na plataforma *online*, onde os gastos eram muitos altos devido ao alto custo de compra e manutenção de servidores, além do fato de que hardwares e outras tecnologias envolvidas nesse processo são propensos a ficarem obsoletos rapidamente, contribuindo muito para o aumento dos custos.

Não é apenas como modo de distribuição de recursos que a computação em nuvem tem se tornado destaque, mas também no que se refere a bancos de dados em geral. No final de 2010 a empresa Embarcadero [15] realizou uma pesquisa sobre tendências, principais iniciativas, ferramentas atuais e desafios de banco de dados realizado com profissionais de banco de dados de grandes organizações. Esta pesquisa destacou que as tecnologias de banco de dados em nuvem e a virtualização terão o maior impacto na indústria de banco de dados, com 34% e 25% das respostas dos entrevistados, respectivamente. Isso indica uma previsão de adoção constante de tecnologias de banco de dados em nuvens.

A replicação de dados é utilizada na computação em nuvem, e consiste em fazer cópias de arquivos já existentes. Quando a replicação de dados começou a ser utilizada, o principal objetivo de sua utilização era a tolerância a falhas, que ocorriam rotineiramente nos sistemas de computação. As réplicas que ficavam no sistema de arquivos eram salvas como cópia para *backup* assim que o arquivo original mudasse de versão, e muitas vezes as réplicas que foram criadas não eram utilizadas em momento algum, sendo praticamente esquecidas durante toda a execução dos processos na nuvem, mas seus custos de armazenamento elevados eram facilmente notados [32]. Entretanto, com os avanços das tecnologias de replicação de dados, as réplicas então sendo utilizadas para permitir que haja maior disponibilidade aos arquivos criados, e assim aumentar o paralelismo nas transições e permitir maior capacidade de leitura e escrita de dados na nuvem na qual a replicação foi empregada.

A utilização das réplicas tem sido estudada cada vez mais para permitir que o processamento ocorra de maneira mais rápida, e grandes resultados têm sido obtidos, como ocorre em [22], onde a política de replicação de dados permitiu que o tempo de resposta às requisições de dados ficasse muito mais rápido.

1.1 Problema

O problema que será abordado nesta pesquisa é que muitos sistemas de armazenamento de dados em nuvem que geram cópias para backup não contam com políticas que utilizem de forma inteligente as cópias, resultando no desperdício de espaço de armazenamento com a função esporádica de recuperação de dados.

1.2 Objetivos

O objetivo dessa pesquisa é desenvolver uma política de replicação de dados que seja capaz de gerenciar, localizar e distribuir as réplicas criadas com eficiência, de modo que o desempenho geral da nuvem seja superior ao alcançado sem a utilização dessas políticas. Os objetivos específicos são:

- Distribuir as operações de leitura e escrita nos computadores de acordo com os arquivos que estão em sua posse.
- Balancear as tarefas entre computadores conforme sua capacidade de processamento.

1.3 Estrutura do Trabalho

Este trabalho está estruturado de acordo com a seguinte sequência:

- **Capítulo 2** – Contém as principais definições sobre computação em nuvem, apresentando as características essenciais de qualquer nuvem computacional, e apresenta os modelos de serviços providos e como são implantadas essas nuvens. Além disso, nesse capítulo são feitas considerações a respeito do mercado da computação em nuvem.
- **Capítulo 3** – Apresenta a replicação de dados dentro do contexto dos bancos de dados em nuvem, e para exemplificar como podem ocorrer as replicações nos sistemas de arquivos distribuídos dos gerenciadores de banco de dados NoSQL, alguns dos mais conhecidos gerenciadores serão apresentados e suas respectivas políticas de replicação serão explicitadas.
- **Capítulo 4** – É o que apresenta a tecnologia utilizada no Apache Hadoop e além disso descreve as estruturas que fazem parte do seu sistema de arquivos e como cada uma delas se comporta para obter a localização dos dados, efetuar o gerenciamento das réplicas e distribuir as réplicas que são inseridas.
- **Capítulo 5** – Elucida os conceitos fundamentais de Biologia Molecular e Bioinformática que são utilizados neste trabalho.
- **Capítulo 6** – Realiza as comparações entre algumas políticas de replicação de dados analisando seu desempenho a partir de um banco de dados de bioinformática.
- **Capítulo 7** – Neste capítulo, finalmente, são apresentados as conclusões e os trabalhos futuros.

Capítulo 2

Computação em Nuvem

Este capítulo apresenta os conceitos fundamentais relacionados à computação em nuvem para o desenvolvimento deste trabalho. Na Seção 2.1 tem-se uma contextualização geral sobre o termo computação em nuvem. Na Seção 2.2 as características essenciais da computação em nuvem são abordadas. Na Seção 2.3 os modelos de serviço são apresentados e na Seção 2.4 há uma breve explicação de cada um dos modelos de implementação.

2.1 Contextualização

O termo computação em nuvem é de origem incerta [10], contudo a expressão ganhou destaque entre 2006 e 2007 quando Eric Shmidt, durante uma palestra, chamou de computação em nuvem um novo e emergente modelo, no qual serviços de dados e arquitetura se localizavam em servidores e poderiam ser comprados por demanda. Esse modelo seria, conforme descrito por Brewer [11], definido pelas duas últimas letras do teorema CAP (disponibilidade e consistência), que foi descrito no Capítulo 1, e está ilustrado na Figura 2.1.

Figura 2.1: Teorema CAP proposto por Brewer [11].

Funcionalmente, a computação em nuvem é desenvolvida como camada de abstração de onde os dados ficam de fato armazenados e são processados, a qual permite que seus usuários tenham amplo acesso a seus serviços e recursos computacionais, que é descrita pelo NIST (*National Institute of Standards and Technology*) como:

> "Um modelo que possibilita acesso, de modo conveniente e sob demanda, a um conjunto de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicações e serviços), que podem ser rapidamente adquiridos e liberados com o mínimo de esforço gerencial ou interação com o provedor de serviços" [23].

Embora a definição de computação em nuvem possa parecer complexa, num primeiro instante, ela foi projetada para que seus usuários pudessem utilizá-la da maneira mais simples quanto possível. A infraestrutura projetada para suportar o ambiente em nuvem normalmente é constituída de diversas máquinas físicas de baixo custo que estão dispostas em uma rede, e podem ser divididas entre máquinas virtuais (processamento) e armazenamento, formando os conhecidos *data centers*, como mostrado na Figura 2.2.

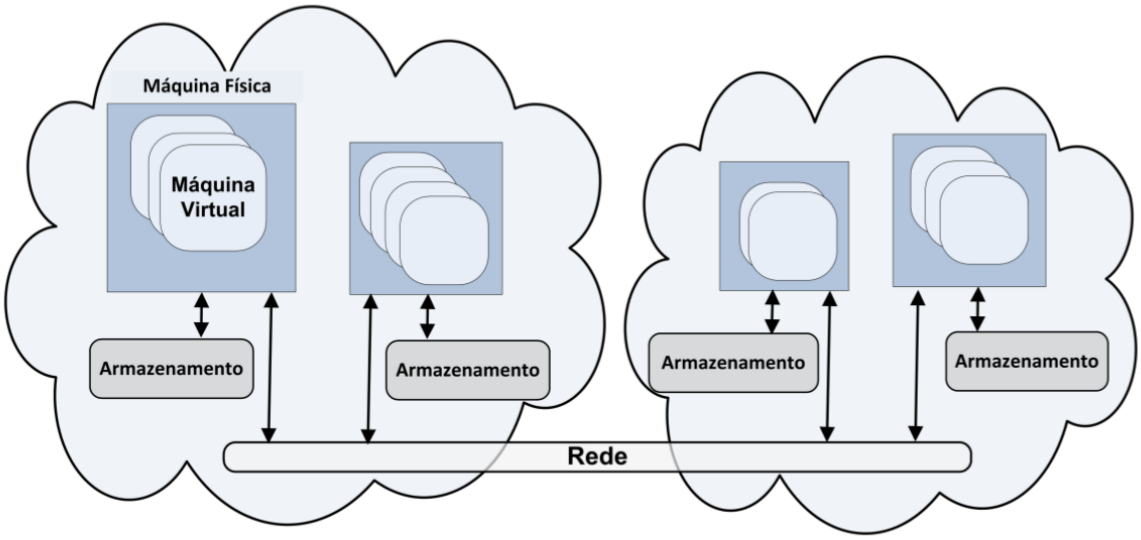

Figura 2.2: Ambiente em Nuvem [23].

2.2 Características Essenciais

De acordo com NIST, a computação em nuvem apresenta cinco características essenciais [23]: *self-service* sob demanda, amplo acesso, *pooling* de recursos, elasticidade rápida e serviço medido, como é possível observar na Figura 2.3. Cada uma dessas características é descrita a seguir.

Self-service sob Demanda

Capacidade do usuário que está utilizando o sistema de adquirir unilateralmente recurso computacional de forma a satisfazer suas necessidades momentâneas ou permanentes sem a necessidade da interação humana para contactar os provedores de serviços. Assim, o usuário pode contar com a qualidade de ter um serviço automático de gestão de recursos computacionais que colocará ao seu dispor toda a capacidade de armazenamento, tempo de processamento ou qualquer outro recurso que o usuário necessite.

Amplo Acesso

Os recursos são compartilhados através da rede, seja ela a Internet. Para garantir a maior acessibilidade possível os mecanismos de disponibilização dos recursos devem ser padronizados de forma a permitir também o uso de *thin clients*. *Thin clients* são máquinas clientes que contam com um servidor de aplicativos que realiza as tarefas mais relevantes de sua lógica interna e possuem o mínimo de hardware e software na sua arquitetura, tais como celulares, *smartphones*, *tables* e PDAs.

Pooling de Recursos

Os recursos computacionais do provedor devem ser organizados em um *pool* de maneira que possa servir a múltiplos usuários em um modelo *multi-tenant* ou multi-inquilino, com diferentes recursos físicos e virtuais, que são dinamicamente atribuídos e ajustados de acordo com a demanda dos usuários e cada configuração de serviço. Os usuários em questão não precisam conhecer precisamente a localização física dos recursos computacionais, podendo apenas indicar onde o recurso estaria em um nível mais alto de abstração, seja por país, estado, *data center* ou nome da rede. A principal importância dessa característica é não deixar para o usuário a necessidade de conhecer os detalhes de implementação, criando uma interface que abstraia da comunicação com o usuário pormenores que não contribuem para a qualidade do serviço prestado.

Elasticidade Rápida

Recursos podem ser adquiridos de forma rápida e elástica, em alguns casos automaticamente, caso haja a necessidade de escalar com o aumento da demanda, e liberados, na retração dessa demanda, ou seja, o provedor deve ser altamente flexível para se adequar às oscilações de demanda por recurso. Para os usuários, os recursos disponíveis para uso parecem ser ilimitados e podem ser adquiridos em qualquer quantidade e a qualquer momento, para isso também é necessário que o provedor mantenha-se informado a respeito do dinamismo da atividade de seus clientes a fim de que possa prever a demanda por recursos antes que estes se esgotem.

Serviço de Medida

Sistemas em nuvem, automaticamente, controlam e otimizam o uso de recursos por meio de uma capacidade de medição. A automação é realizada em algum nível de ab-

stração apropriado para o tipo de serviço, tais como armazenamento, processamento, largura de banda e contas de usuário ativas. O uso de recursos pode ser monitorado e controlado, possibilitando transparência para o provedor e o usuário do serviço utilizado.

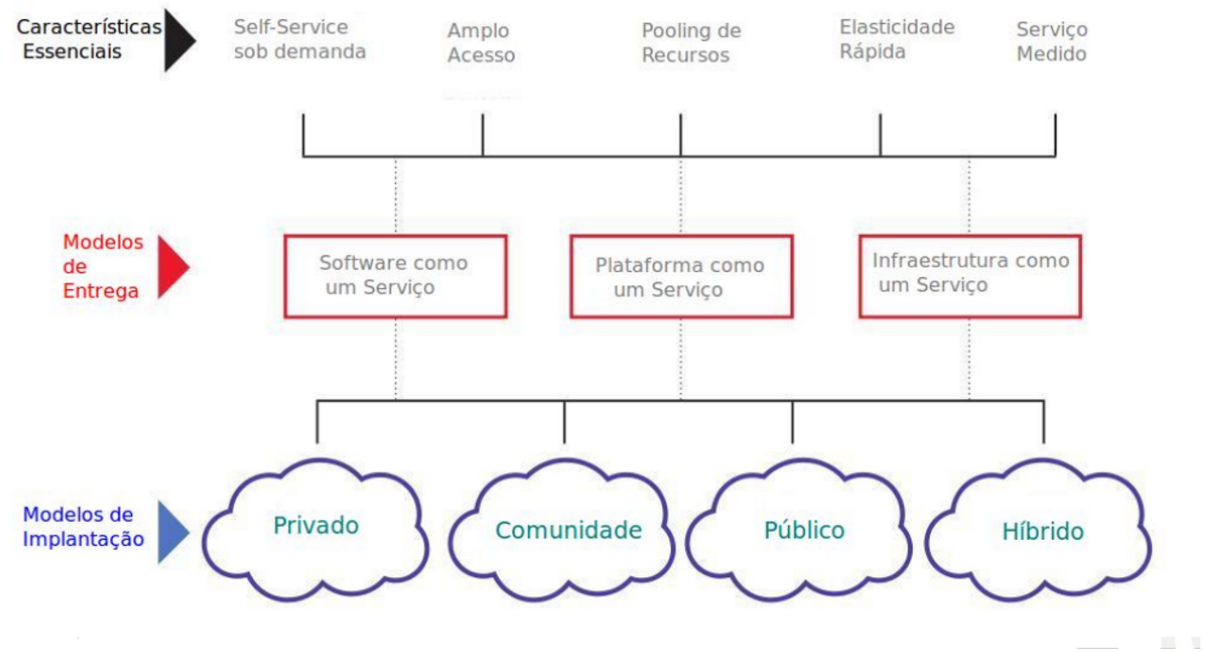

Figura 2.3: Modelo NIST de Computação em Nuvem [23].

2.3 Modelos de Serviço

Ainda de acordo com o NIST existem três modelos de serviço [23], os quais são: Software como um Serviço (SaaS), Plataforma como Serviço (PaaS) e Infraestrutura como Serviço (IaaS). Essas características são explanadas em seguida e ilustradas na Figura 2.4.

Software como um Serviço (SaaS)

O modelo de SaaS, proporciona sistemas de software com propósitos específicos que são disponíveis para os usuários por meio da Internet, e acessíveis a partir de vários dispositivos do usuário por meio de uma interface *thin client* como um navegador Web. No SaaS, o usuário não administra ou controla a infraestrutura subjacente, incluindo rede, servidores, sistema operacional, armazenamento ou mesmo as características individuais da aplicação, exceto configurações específicas. Como exemplos de SaaS destaca-se os serviços de *Customer Relationship Management* (CRM) da Salesforce [29] e o Google Docs [17].

Plataforma como Serviço (PaaS)

O modelo de PaaS fornece sistema operacional, linguagens de programação e ambientes de desenvolvimento para as aplicações, auxiliando a implementação de sistemas de software. Assim como no SaaS, o usuário não administra ou controla a infraestrutura subjacente, mas tem controle sobre as aplicações implantadas e, possivelmente, as configurações de aplicações hospedadas nesta infraestrutura. Google App Engine [13] e Microsoft Azure [24] são exemplos de PaaS.

Infraestrutura como Serviço (IaaS)

A IaaS torna mais fácil e acessível o fornecimento de recursos, tais como servidores, rede, armazenamento e outros recursos de computação fundamentais para construir um ambiente de aplicação sob demanda, que podem incluir aplicativos de hardware. Em geral, o usuário não administra ou controla a infraestrutura da nuvem, mas tem controle sobre os sistemas operacionais, armazenamento, aplicativos implantados e, eventualmente, seleciona componentes de rede, tais como *firewalls*. O *Amazon Elastic Compute Cloud* [3] e o *Eucalyptus* [16] são exemplos de IaaS.

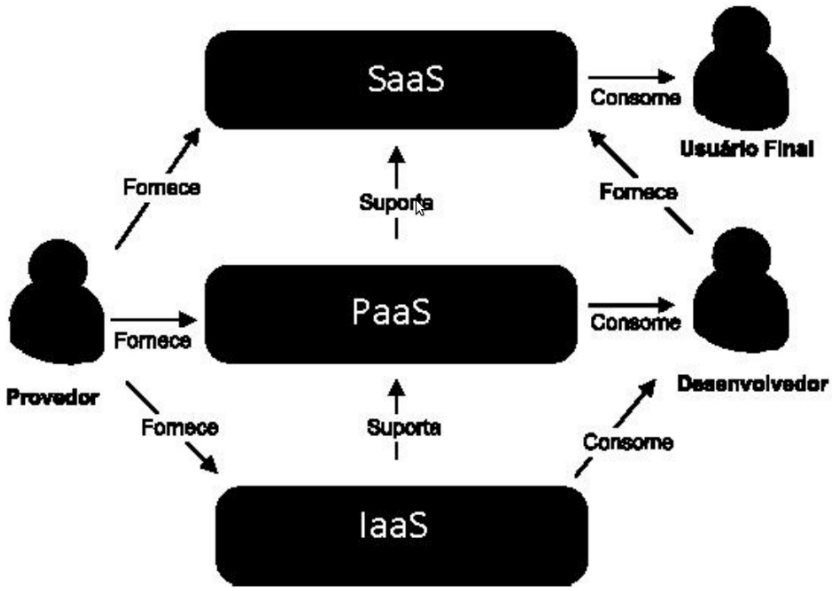

Figura 2.4: Modelos de serviço e papéis dos atores [23].

2.4 Modelos de Implantação

Quanto ao acesso e à disponibilidade, há diferentes tipos de modelos de implantação para os ambientes de computação em nuvem. A restrição ou abertura de acesso depende do processo de negócios, do tipo de informação e do nível de visão desejado [23]. Os dos modelos descritos pelo NIST são:

- **Nuvem privada:** A infraestrutura de nuvem é utilizada exclusivamente por uma organização, sendo esta nuvem local ou remota e administrada pela própria empresa ou por terceiros.

- **Nuvem pública:** A infraestrutura de nuvem é disponibilizada para o público em geral, sendo acessado por qualquer usuário que conheça a localização do serviço.

- **Nuvem comunitária:** Fornece uma infraestrutura compartilhada por uma comunidade de organizações com interesses em comum.

- **Nuvem híbrida:** A infraestrutura é uma composição de duas ou mais nuvens, que podem ser do tipo privada, pública ou comunidade e que continuam a ser entidades únicas, mas conectadas por meio de tecnologia proprietária ou padronizada que permite a portabilidade de dados e aplicações.

Capítulo 3

Replicação de Dados

A replicação de dados é utilizada há muitos anos na computação, mas o seu foco tem variado muito desde que começou a ser empregada. No início, o alvo principal do exercício da replicação de dados era a tolerância a falhas, as quais ocorriam frequentemente nos sistemas de computação. Em seguida, a alta demanda por serviços na web fez com que a replicação fosse utilizada também a fim de propiciar maior disponibilidade aos arquivos mais requeridos. Atualmente, além de cumprir os papéis supracitados, a replicação de dados é utilizada em larga escala para aumentar a velocidade de processamento, já que as operações em disco são o gargalo da computação e o tempo de transmissão de arquivos na rede pode crescer muito dependendo do tamanho do arquivo.

As políticas e procedimentos referentes à replicação de dados estão entre as principais causas da escolha de um banco de dados NoSQL, pois além de tolerarem o particionamento dos dados, proporcionam maior disponibilidade e permitem a configuração do sistema de arquivos de maneira que se adequem ao ambiente de aplicação. Dentre as principais opções de configuração encontradas, destacam-se: a possibilidade de escolha do número de réplicas a serem feitas automaticamente na inserção dos dados; o nível de consistência requerido em detrimento da disponibilidade; e a definição dos caminhos de dados na rede. A política de replicação de dados, portanto, reflete uma característica essencial dos sistema de arquivos distribuídos que são utilizados nos bancos de dados NoSQL, permitindo a cada um deles atender a diferentes demandas dos usuários. Nas seguintes seções deste capítulo estão alguns dos gerenciadores de bancos de dados NoSQL com suas respectivas características de replicação, ao final uma breve comparação entre eles.

3.1 Cassandra

Criado pelo Facebook para o armazenamento de seus dados e, posteriormente, doado para a fundação Apache, o Cassandra passou a ser parte fundamental em muitas empresas de grande porte, tais como: Netflix, eBay, Twitter, Urban Airship, Constant Contact, Reddit, Cisco, OpenX, Digg, CloudKick, Ooyala, e mais outras empresas que possuem grandes servidores ativos, *data centers*. O maior *cluster* do Cassandra conhecido tem acima de 300 TB de dados em mais de 400 máquinas [8].

3.1.1 Características

Cassandra é uma implementação de família de colunas NoSQL que suporta o modelo de dados do Google Big Table e usa aspectos de arquitetura introduzidos pelo Dynamo da Amazon [28]. Alguns dos pontos positivos do Cassandra são:

- Altas escalabilidade e disponibilidade, sem um ponto único de falha;
- Implementação da família de colunas NoSQL;
- Rendimento de gravação muito alto e bom rendimento de leitura;
- Linguagem de consulta semelhante a SQL, a *Cassandra Query Language*, que está disponível desde a versão 0.8;
- Suporte para consulta por índices secundários;
- Consistência ajustável e suporte para replicação;
- Esquema flexível.

Como o Cassandra armazena colunas classificadas por seus nomes, as consultas de trechos do sistema de arquivos são muito rápidas. É importante observar que a classificação de todos os detalhes de um item de dados em uma única linha e o uso de classificações de ordem são as ideias mais importantes por trás do *design* de dados do Cassandra. A maioria dos *designs* de modelo de dados do Cassandra segue essas ideias em alguma forma. O usuário pode usar as ordens de classificação ao armazenar dados e criar índices. Por exemplo, o efeito adverso de anexar registros de data e hora a nomes de coluna é que, como esses nomes são armazenados na ordem classificada, os comentários com nomes de coluna seguidos pelo registro de data e hora são armazenados na ordem em que foram criados, e os resultados de pesquisa teriam a mesma ordem.

Consequentemente, para obter os melhores resultados em um *design* de dados do Cassandra, é necessário que usuários implementem consultas criando índices customizados e utilizando ordens de classificação de coluna e de linha.

3.1.2 Níveis de Consistência das Réplicas

O sistema de replicação proposto pelo gerenciador Cassandra inclui diversos níveis de consistência, que variam de zero (conhecido como dedos cruzados, quando não há verificação de consistência) a todos, controlando o valor de todas as réplicas para serem sempre consistentes.

O número de réplicas padrão do Cassandra é três, para permitir a identificação mínima de um valor inserido incorretamente (dois valores corretos e um incorreto), mas esse valor pode ser alterado facilmente nos arquivos de configuração. A estrutura que recebe os dados gravados é o *commit log*, que é o primeiro local por onde passam os dados e, posteriormente, os dados são encaminhados para uma estrutura de tabela em memória, denominada *memory table*

Para qualquer dada operação de escrita e de leitura, o cliente da aplicação decide quais dos níveis de consistência listados a seguir a operação deve ter.

ONE (1) - Na escrita, garante que o dado foi escrito em um *commit log* e uma tabela de memória de ao menos uma réplica antes de responder ao cliente. Na leitura, o dado será retornado a partir do primeiro nó onde a chave buscada foi encontrada. Essa prática pode resultar em dados antigos sendo retornados, porém, como cada leitura gera uma verificação de consistência em background, consultas subsequentes retornarão o valor correto do dado;

QUORUM (Q) - Na escrita, garante que o dado foi escrito em $fator\_replicao/2+1$, que é determinado pelo arredondamento para baixo do valor do fator de replicação escolhido dividido por dois e acrescido de uma unidade. Na leitura retorna o valor mais recente lido de $fator$
$_replicao/2+1$ réplicas. As réplicas restantes são sincronizadas em *background*;

ALL (N) - Garante que operações de leitura e escrita envolverão todas as réplicas. Assim, qualquer nó que não responda às consultas fará as operações falharem.

Vale ressaltar também outros dois níveis de consistência que não foram mencionados previamente, os quais são:
Local Quorum e *Each Quorum*. Eles são níveis de consistência nas operações que são reservadas para *data centers*. No *Local Quorum*, a escrita deve ser efetivamente concluída no *commit log* e na *memory table* no valor de *quorum* nos nós de replicação do mesmo *data center* do coordenador da operação, evitando o tempo de latência da comunicação entre os diferentes *data centers*. Já no nível *Each Quorum* cada um dos *data centers* deve ter a quantidade mínima do *quorum* para que a operação seja concluída com sucesso, garantindo maior consistência aos dados.

3.2 HBase

Projeto Apache baseado no Google BigTable, o HBase possui uma abordagem de replicação dos dados que envolve um *cluster* mestre que contém todos os servidores regionais chamados HRegionServers. Estes por sua vez efetuam chamadas síncronas aos *clusters* escravos, que irão escrever os dados inseridos ou atualizados paralelamente. O HBase utiliza o HDFS (*Hadoop Distributed File System*), possibilitando que o HBase desfrute das vantagens de armazenamento já desenvolvidas pela própria Apache [5].

3.2.1 Características

Pelo fato de também ser um gerenciador de bancos de dados NoSQL baseado em colunas, o HBase possui muitas características em comum com o Cassandra por parte da organização, mas também possui suas peculiaridades pelo fato da equipe da Apache ter um foco muito diferente do Facebook na implementação de seus produtos. Dentre as principais características do HBase estão:

- Escalabilidade linear e modular;
- Consistência nas leituras e escritas;

- Fragmentação de tabelas automática e configurável;
- Automático suporte a falhas entre servidores;
- Possui extensões que tornam fácil a utilização de programas Java para configurar o acesso dos clientes;
- Cache de bloco e filtros para consultas em tempo-real.

HBase suporta dados não estruturados e parcialmente estruturados [33]. Para fazer isso, os dados são organizados em famílias de colunas. Um registro individual, chamado de "célula" no Hbase, é endereçado com uma combinação de *row key*, *column family*, *cell qualifier*, e *time stamp*. Em oposição ao RDBMS (*Relational Database Management System*), no qual é necessário definir bem sua tabela com antecedência, com o HBase é possível simplesmente nomear uma família de colunas e então permitir que a célula se qualifique para ser determinada a um certo tempo de execução. Isso permite que você seja bastante flexível e suporta uma abordagem ágil de desenvolvimento.

3.2.2 Sistema de Replicação

O sistema de replicação do HBase possui registros (HLogs) dos processos ocorridos, permitindo que se tenha conhecimento integral de cada estágio da replicação e mantendo o servidor sempre informado a respeito de erros e rotinas concluídas.

Os HLogs de cada RegionServer são a base da replicação do HBase, e devem ser mantidos no HDFS por tanto tempo quanto for necessário replicar dados para qualquer *cluster* escravo. Cada RegionServer lê do *log* mais antigo que ele necessite para replicar e mantém a posição corrente dentro do ZooKeeper [9]. O Zookeeper é um serviço que faz o papel de coordenador e gerenciador de quase todas as principais atividades de replicação, para simplificar a recuperação a falhas. A posição do arquivo pode ser diferente em cada *cluster* escravo, o mesmo ocorre para consultas de HLogs para processar, como mostrado na Figura 3.1, que ilustra o momento de uma replicação no HBase. Assim que o HRegionServer efetua a leitura dos *logs* referentes à replicação, há uma chamada síncrona aos *clusters* escravos para que haja replicação de determinado dado.

Os *clusters* que participam da replicação podem ser de tamanhos assimétricos e o *cluster* mestre irá fazer o "melhor esforço" para balancear o fluxo de replicações entre os *clusters* escravos confiando na randomização.

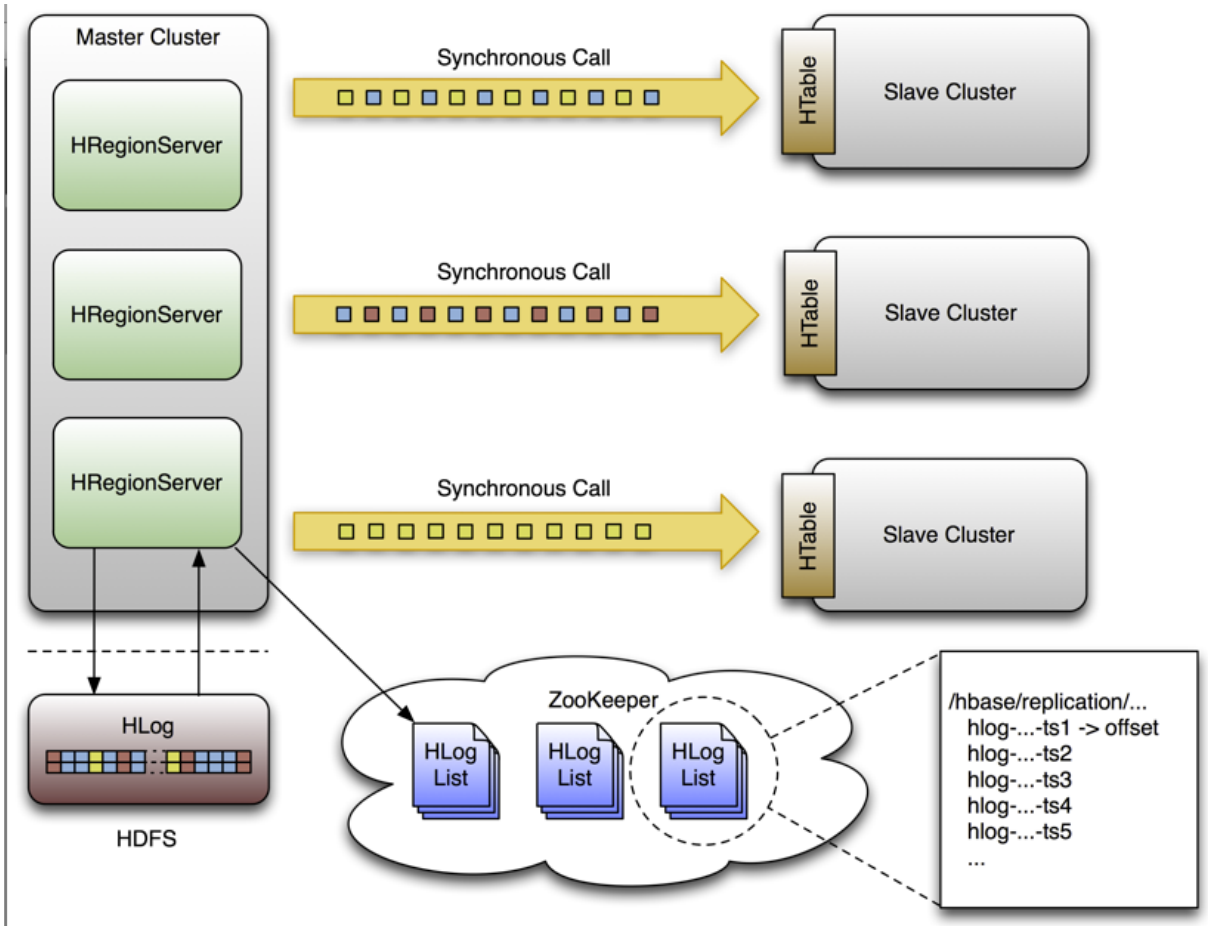

Figura 3.1: Replicação do HBase [5].

3.3 MongoDB

MongoDB (do "humongous") é uma aplicação de código aberto, de alta performance, sem esquemas, orientado a documentos. Foi escrito na linguagem de programação C++. Além de orientado a documentos, é formado por um conjunto de documentos JSON. Muitas aplicações podem, dessa forma, modelar informações de modo muito mais natural, pois os dados podem ser aninhados em complexas hierarquias e continuar a ser indexáveis e fáceis de buscar. O desenvolvimento do MongoDB começou em outubro de 2007 pela 10gen [1]. A primeira versão pública foi lançada em fevereiro de 2009 [25].

3.3.1 Conjunto de Réplicas

MongoDB suporta replicação assíncrona de dados entre servidores para evitar falhas e obter redundância. Durante a replicação, o MongoDB mantém um ou mais membros secundários ativos para leituras e, ocasionalmente, um terciário para recuperação no caso de erro durante a operação. O membro primário, ou *master*, efetua a operação de escrita exclusivamente em um determinado momento, para que depois o valor seja atualizado nos demais membros [26]. Este comportamento é chamado de consistência eventual, porque

o estado dos membros secundários e de recuperação irão eventualmente refletir o estado do primário e o MongoDB não pode garantir a consistência estrita para membros não-primários, excepcionalmente configurando o cliente e o *driver* do MongoDB para garantir que as operações obtiveram êxito em todos os membros antes de completar a leitura como bem sucedida. A Figura 3.2 ilustra o esquema de replicação, a seta tracejada corresponde à troca de informações entre o membro primário e o de recuperação, e a outra seta refere-se à comunicação entre o membro primário e o membro secundário.

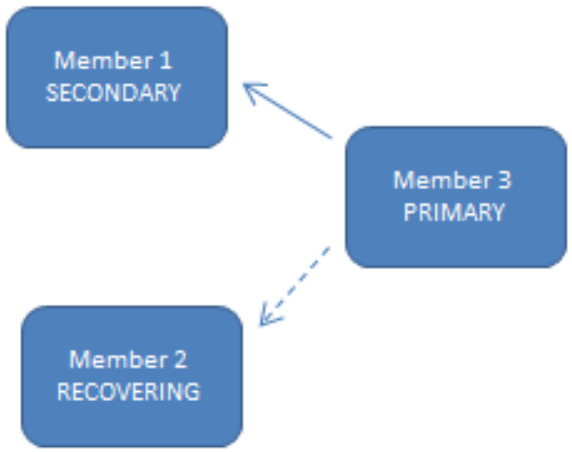

Figura 3.2: Replicação do MongoDB Usando Nós de Consistência Eventual [26].

O Replica Set, o conjunto de réplica do MongoDB, permite que os membros secundários da operação sejam configurados de diversas maneiras, tais como apresentadas a seguir.

Membros Unicamente Secundários

A configuração de nó unicamente secundário previne que um determinado membro secundário ocasionalmente se torne um membro primário no caso de falha. É possível configurar qualquer quantidade de membros do conjunto.

Por exemplo, é possível determinar que a configuração de todos os membros do conjunto de replicação localizados fora do *data center* principal como unicamente secundários para previnir que estes membros se tornem primários. Para configurar um membro como unicamente primário basta escolher a prioridade dele como zero no arquivo de configuração. Qualquer membro com a prioridade igual a zero nunca irá buscar a eleição e não pode se tornar primário em qualquer situação.

Membros Ocultos

Membros ocultos são parte do conjunto de réplica, mas não podem ser primários e são invisíveis para as aplicações clientes. Entretanto, membros ocultos votam para eleger o primário.

Membros ocultos são ideais para instantes que terão padrões de uso significantemente diferentes dos outros membros, e requerem distinção do tráfego normal. Tipicamente,

membros ocultos fornecem relatórios *backups* dedicados, e testes e suporte para operação do tipo *read-only* dedicados.

Membros Atrasados

Membros atrasados copiam e aplicam as operações decorrentes do *oplog*, o registro de operações, do membro primário com um atraso especificado. Se um membro tem um atraso de uma hora, então a última entrada no *oplog* deste membro não será mais recente do que uma hora atrás, e o estado dos dados para o membro irá refletir o estado do conjunto uma hora mais cedo.

Membros atrasados podem ajudar na recuperação de diversos tipos de erros humanos. Tais erros podem incluir bancos de dados excluídos inadvertidamente ou atualizações fracassadas de aplicações. Assim devem ser considerados os seguintes fatores na hora de determinar a quantidade de atraso a aplicar ao escravo:

- Assegure-se de que o comprimento do *delay* é igual ou maior do que sua janela de manutenção;
- O tamanho do *oplog* é suficiente para guardar mais do que o número de operações que ocorrem tipicamente naquele período de tempo;
- Membros atrasados devem ter a prioridade fixada em zero para previní-los de se tornarem membros primários nos seus conjuntos de réplica. Esses membros também deveriam ser ocultos para prevenir que alguma aplicação veja ou consulte este membro.

Árbitros

Árbitros são instâncias especiais que não mantêm uma cópia do dado, e assim não podem se tornar primários. Árbitros existem somente para participar em eleições.

Por causa de seus requisitos mínimos de sistema, é possível implantar um árbitro num sistema com outra carga de trabalho, tal como um servidor de aplicativos ou um membro de monitoramento.

Os árbitros nunca recebem o conteúdo de qualquer coleção, mas têm as seguintes interações com o resto do conjunto de réplicas:

- Troca de credenciais que autenticam o árbitro com o conjunto de réplicas. Todos os processos do MongoDB dentro de um conjunto de réplica usam arquivos-chave;
- O MongoDB somente transmite as credenciais de autenticação numa troca criptograficamente segura. E não criptografa nenhuma outra troca;
- Mudança de dados de configuração do conjunto de réplicas e de votos. Esses não são criptografados.

Membros Não-Votantes

É possível escolher mudar o número de votos que cada membro tem nas eleições para o membro primário. Em geral, todos os membros deveriam ter um voto para previnir laços intermitentes, *deadlocks* ou que os membros errados se tornem primários. É recomendável

utilizar as propriedades do conjunto de réplica para controlar quais membros são mais desejáveis de se tornarem primários.

3.4 Dynamo

Para permitir um simples *design*, robustez e alta performance, a Amazon limitou as capacidades de relatório - o dado é acessado pela chave primária somente. Para implementar uma requisição de alta disponibilidade, o Dynamo replica cada par chave-valor entre múltiplas máquinas em diferentes armazenadores de dados. E para ter certeza de que a replicação não iria impactar a performance, a Amazon decidiu que eles poderiam conviver sem a consistência - as requisições para o sistema podem algumas vezes retornar dados antigos ou versões contraditórias do dado - para os requerimentos da Amazon, isto é muito melhor do que falhas ou atrasos [30].

O equilíbrio entre consistência e performance é muito explicitamente tratada nas especificações da Amazon - o sistema terá muitas versões do dado, uma requisição pode apanhar qualquer número de réplicas e pode não pegar a versão com as mais recentes atualizações. Os desenvolvedores de aplicativos têm que construir aplicações que podem suportar esta situação e mesclar várias versões do dado na informação que eles querem.

Para evitar perder dados, o sistema usa um sistema do tipo *quorum*. Aqui estão três parâmetros para escolher a replicação: quantas vezes cada par chave-valor deve ser replicado (N), quantos nós deveriam participar em uma escrita bem sucedida (W) e quantos nós devem participar de uma leitura com sucesso.

Quando W é igual a 3, ao menos 3 nós devem assinalar que a escrita foi completada com sucesso antes da aplicação receber um sinal de confirmação do armazenamento de dados. Um nó coordenador deve atentar para escrever N nós, mas uma vez que W nós aprovem a escrita, esta é considerada permanente. Segue que, para perder uma atualização é necessário que ao menos W nós falhem simultaneamente depois de uma atualização e provavelmente mais do que W, porque ele é apenas o valor mínimo. Enquanto ao menos um dos nós está ativo, o dado não está perdido.

A ideia é escolher para o nível de replicação um número alto o suficiente para satisfazer as necessidades de durabilidade, mas não tão alto, para prevenir muita latência na atualização dos dados. O usuário também pode querer dividir os nós entre, no mínimo, dois *data centers* para reduzir a chance de perder todos os nós de uma única vez, por isso o sistema também implementa suas operações de divisão do *cluster* [34].

Considerando que este sistema não permite *backups*, o número de nós replicados deve ser realmente alto o bastante. O número de replicações padrão do Dynamo é três, e a Amazon ressalta que não perder uma única atualização é vital para o negócio, pois uma atualização pode ser a venda de um item.

O resultado dessa política de replicação é mecanismo de replicação dinâmica que copia os dados em múltiplos *hosts* num fator N, isso produz um sistema no qual cada nó é responsável pelas réplicas da região compreendida por seus N-1 predecessores, assim como apresentado na Figura 3.3.

A lista dos nós que são responsáveis por uma chave particular é chamada lista de preferências. Então em cada nó no sistema é possível determinar quais nós estariam na lista para uma chave particular. Para prevenir falhas nos nós, as listas de preferências

podem conter mais do que N nós, fazendo uso de nós virtuais, ou seja, um nó pode mais do que uma das N primeiras posições.

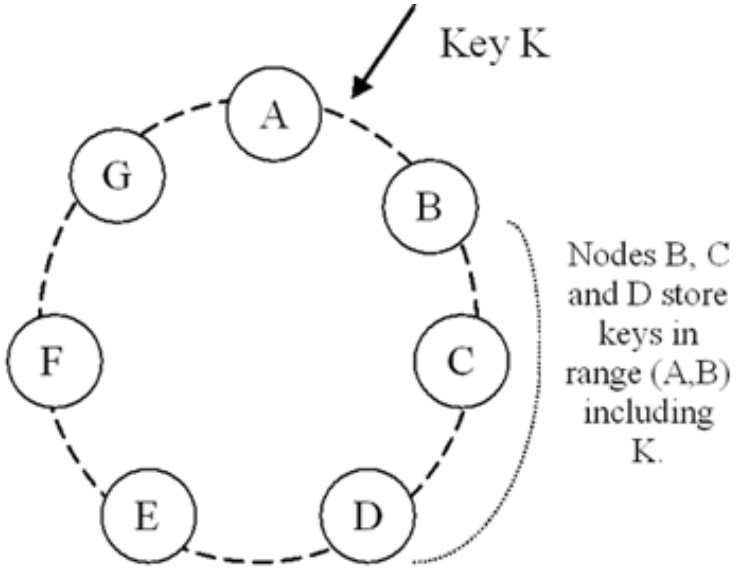

Figura 3.3: Replicação nos Nós Posteriores com N = 4 [4].

3.5 Comparação das Arquiteturas

Para expor de maneira mais clara e objetiva as características de cada um dos gerenciadores NoSQL foi feita a tabela 3.1.

Tabela 3.1: Comparação das Características.

Gerenciador	Chamada	Servidores
Cassandra	Síncrona	Multiservidor
HBase	Síncrona	Servidor Regionalizado
MongoDB	Síncrona	Multiservidor
Dynamo	Síncrona	Descentralizados

Pelo fato de estarem numa camada muito acima das políticas de replicação, como pode-se notar na figura 3.4, os gerenciadores NoSQL não permitem que os usuários configurem as suas próprias políticas de replicação e as abstrações entre as camadas fazem com que a implementação de sistemas de arquivos distribuídos se tornem muito menos eficientes.

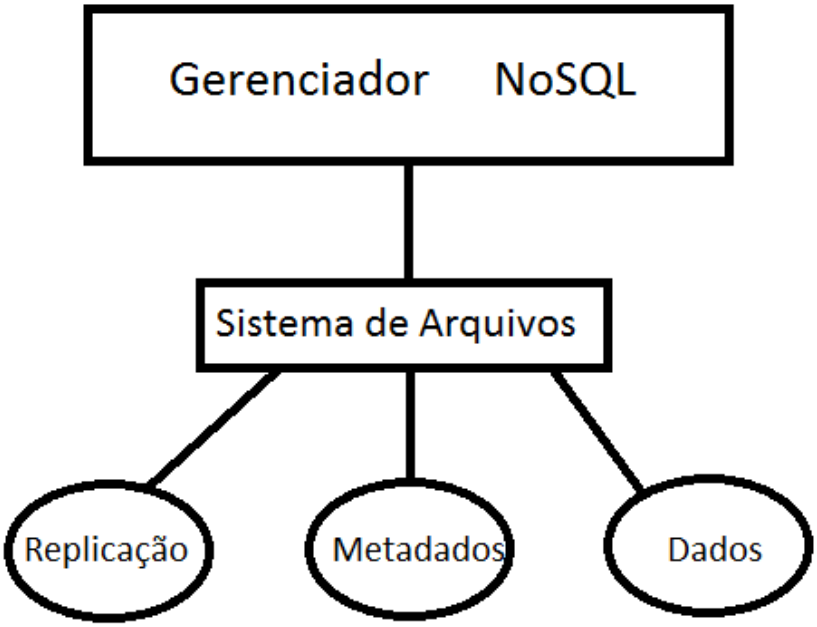

Figura 3.4: Localização da Política de Replicação no Ambiente em Nuvem

Capítulo 4

Ambiente em Nuvem Utilizando Apache Hadoop

Nas seções deste capítulo haverá uma breve introdução à história do Apache Hadoop, seguida de uma seção com a visão geral do sistema de arquivos adotado, para enfim explicar detalhadamente cada uma das estruturas que compõem o sistema de arquivos e definir como cada uma delas participa dos estágios de replicação, inserção e exclusão de arquivos.

4.1 História do Apache Hadoop

A história do Apache Hadoop começa entre 2002 e 2004, quando a Yahoo! detectou a necessidade do desenvolvimento de um mecanismo que permitisse a pesquisa na web, que fosse distribuído e possibilitasse processamento baseado em *sort* e *merge*. A primeira versão foi demonstrada em apenas quatro nós e percorreu cerca de 100 milhões de páginas da web, com alto custo e com escalabilidade muito baixa [14].

No ano 2004, a Google lançou seu artigo sobre o Google File System (GFS) [19] e o MapReduce [18], as idéias publicadas no artigo ajudou o grupo de desenvolvimento do Hadoop, inicialmente composto apenas por Cutting e Cafarella, a resolver alguns dos problemas relacionados à programação e execução para o Hadoop. Isso permitiu que o Hadoop pudesse ser executado em 20 nós e foi escalável até a ordem dos 100 milhões de páginas da web, mas ainda muito distante de poder ser utilizado na escala de toda a web [14].

A partir de 2006, o Hadoop passou a ter uma equipe de desenvolvimento para o projeto. A doação do projeto Hadoop para a Apache permitiu que mais desenvolvedores pudessem participar do desenvolvimento e testar as funcionalidades, assim, finalmente o Hadoop chegou ao nível de escalabilidade da web no começo de 2008.

4.2 Visão Geral do Hadoop

O Sistema de Arquivos Distribuídos do Hadoop (HDFS) é um sistema de arquivos distribuído projetado para ser executado em computadores comuns [35]. Ele tem muitas similaridades com sistemas distribuídos já existentes. Entretanto, as diferenças com outros

sistemas de arquivos são significantes. Entre as características do Hadoop, encontram-se a alta tolerância a falhas e o fato de ser desenvolvido para ser implantado em computadores de baixo custo, facilitando a expansão do sistema e reduzindo os custos para utilização.

O HDFS é altamente configurável e possui um ajuste padrão bem adequado para muitas situações. Na maior parte das vezes, as *configurações* só precisam ser alteradas para *clusters* muito largos, ou com propósitos específicos. O fato do Hadoop ter sido desenvolvido em Java possibilita sua execução independentemente da plataforma utilizada e do sistema operacional, inclusive aumentando a compatibilidade de suas operações entre diversos tipos de máquinas.

O HDFS também permite comandos do modelo *shell* para interagir com o sistemas de arquivos diretamente. Esses comandos permitem alterar as permissões, criar e excluir pastas e arquivos, utilizar um modo de segurança para a manutenção, além de muitas outras utilidades.

Nos seus 10 anos de existência, o Hadoop tem sido uma resposta eficiente para os motores de busca da Yahoo, sendo uma plataforma de propósito geral de computação que se posiciona para ser o fundamento para as próximas gerações de aplicações de base de dados.

Depois que o Hadoop se tornou um projeto da Apache, popularizou-se rapidamente pela Internet, principalmente pelo fato de ser a única ferramenta de código aberto que permitia a manipulação de uma grande base de dados. Hoje, muitos sites que utilizam largas quantidades de dados usam o Hadoop, tais como, o Facebook, o eBay, a Amazon, o Twitter, entre outros.

Estima-se que o mercado proporcionado pelo Hadoop irá crescer cerca de 60% ao ano até atingir a marca de 13,9 bilhões de dólares em 2017 [21].

4.3 Arquitetura do Sistema de Arquivos

A arquitetura do Hadoop divide-se entre as partes do sistema de arquivos. Na parte física encontram-se as configurações de hardware utilizadas e as transferências utilizando a rede da nuvem. O sistema também é constituído de *Namenode*, *Datanode*, replicação de dados, módulo de persistência, metadados, organização de dados, acessibilidade e gerência de reinvindicação de espaço, além de outras funcionalidades. A Figura 4.1 ilustra a arquitetura do sistema de arquivos entre dois *racks* de computadores, no qual dois clientes fazem requisições, um de leitura e outro de escrita, nota-se que no momento da escrita ocorre a replicação do valor em outro bloco do sistema de arquivos, revelando a localização da réplica do bloco utilizado. Também é possível notar que durante as operações o *Namenode* guarda os metadados e os *Datanodes* armazenam os blocos de dados persistentes no sistema de arquivos. A próxima subseção aborda as características do sistema de arquivos. E a partir da seção 4.4 cada uma das partes do sistema de arquivos será apresentada separadamente.

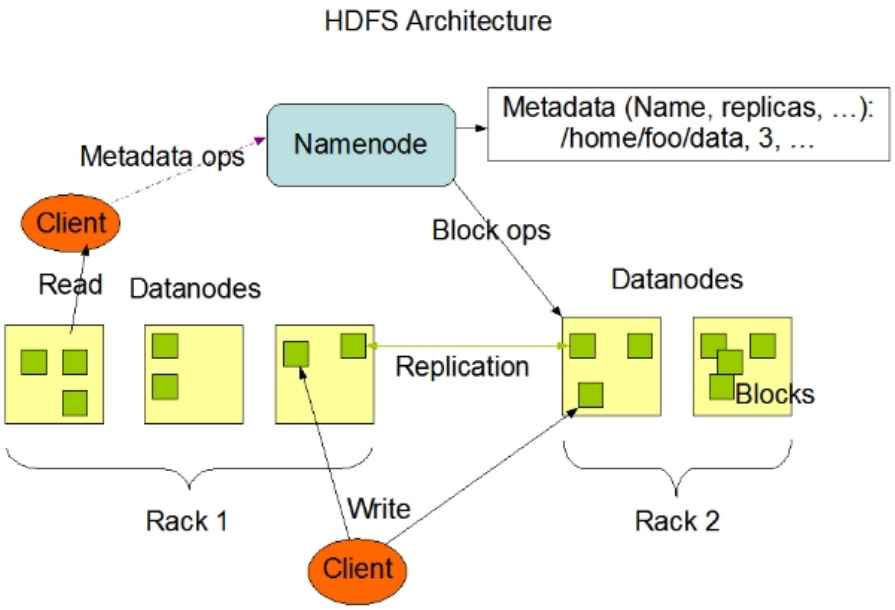

Figura 4.1: Arquitetura do Hadoop [35].

4.3.1 Características

Para o Hadoop a falha de hardware é mais uma regra do que uma exceção. Uma instância do HDFS pode consistir de centenas ou milhares de máquinas, cada uma delas armazenando uma parte dos dados do sistema de arquivos. O fato de que alí estão um grande número de componentes e que cada componente tem uma probabilidade não-trivial de falha significa que alguns componentes do HDFS estão sempre fora de funcionamento. Portanto, a detecção de falhas é rápida e automática, e a recuperação delas é um objetivo central da arquitetura do HDFS.

Aplicações que são executadas no HDFS necessitam de acesso de transmissão para seus conjuntos de dados. Elas não são aplicações de propósito geral que tipicamente são executadas em sistemas de arquivos de propósito geral. O HDFS é formulado mais para processamento em *batch* do que uso interativo pelos usuários. A ênfase está mais em alto *throughput* de acesso aos dados do que em baixa latência no acesso aos dados.

Aplicações que são executadas no HDFS têm um grande conjunto de dados. Um típico arquivo armazenado no HDFS tem um tamanho que vai de *gigabytes* a *terabytes* de comprimento. Então, o HDFS é ajustado para suportar arquivos grandes. Ele deveria prover alta largura de banda para agregação de dados. Ele deveria suportar dezenas de milhões de arquivos em uma única instância.

As aplicações do HDFS necessitam de um modelo de acesso para arquivos do tipo escreva-uma-vez-leia-várias. O arquivo, uma vez criado, escrito e fechado não precisa ser modificado. Essa hipótese simplifica questões de coerência de dados e habilita o alto *throughput* de acesso de dados. Uma aplicação Map/Reduce ou uma aplicação de rastreação web se encaixa perfeitamente neste modelo. Há um plano para suportar escritas anexas para arquivos no futuro.

Uma requisição de computação por uma aplicação é muito mais eficiente se essa é executada próxima aos dados que ela opera. Isso é, especialmente, verdadeiro quando o tamanho do volume de dados é grande. Isso miniminiza o congestionamento da rede e aumenta o *throughput* global do sistema. A hipótese é que, frequentemente, é melhor migrar a computação para mais perto de onde os dados estão localizados do que mover os dados para onde a aplicação está sendo executada. O HDFS provê interfaces para aplicações para movê-las para mais perto de onde o dado está localizado.

O Hadoop foi formulado para ser facilmente portável de uma plataforma para outra. Para isso seu código foi escrito em Java e suas configurações são independentes da plataforma em uso. A adoção dessas características facilita a utilização amplamente difundida do Hadoop e o torna uma plataforma de sistema de arquivos distribuídos viável para um largo conjunto de aplicações.

4.4 *NameNode* e *DataNodes*

O HDFS tem uma estrutura do tipo Mestre/Escravo. Um *cluster* HDFS consiste de um único *NameNode*, um servidor mestre que gerencia o sistema de arquivos e regulamenta o acesso a arquivos pelos clientes. Além disso, há também um certo número de *DataNodes*, usualmente um por nó do *cluster*, o qual gerencia o armazenamento anexado aos nós nos quais eles são executados. O HDFS apresenta um sistema de arquivos e permite que os dados dos usuários sejam armazenados em arquivos. Internamente, um arquivo é divido em um ou mais blocos, e esses blocos são armazenados em um conjunto de *DataNodes*. O sistema de arquivos do *NameNode* executa operações como abrir, fechar e renomear arquivos e diretórios. Isso também determina o mapeamento de blocos para o *DataNode*. O *DataNode* é responsável por servir requisições de leitura e escrita dos clientes do sistema de arquivos. Como pode ser observado na figura 4.1.

O *NameNode* e o *DataNode* são componentes de software projetados para serem executados em máquinas comuns. Essas máquinas tipicamente executam um sistema operacional (OS) GNU/Linux. O HDFS é feito usando a linguagem Java. Assim, qualquer máquina que suporte Java pode executar o software do *NameNode* ou do *DataNode*. A utilização da linguagem de alta portabilidade Java significa que o HDFS pode ser implantado numa grande variedade de máquinas. Uma implantação típica tem uma máquina dedicada que executa apenas o software do *NameNode*. Cada uma das outras máquinas no cluster executa uma instância do software do *DataNode*. A arquitetura não impede que sejam executados múltiplos *DataNodes* na mesma máquina, mas em um desenvolvimento real, esse é raramente o caso.

A existência de um único *NameNode* no *cluster* simplifica grandemente a arquitetura do sistema. O *NameNode* é um mediador e repositório para todos os metadados do HDFS. O sistema é desenvolvido de tal maneira que o usuário nunca passa através do *NameNode*, ou seja, ele só tem acesso aos dados, e nunca pode promover alterações na arquitetura do HDFS.

4.5 O Espaço Nominal do Sistema de Arquivos

O HDFS suporta uma organização de arquivos hierárquica tradicional. O usuário ou uma aplicação podem criar diretórios e armazenar arquivos dentro desses diretórios. A hierarquia do espaço nominal do sistema de arquivos é similar a maior parte dos outros sistemas de arquivos existentes. Alguém pode criar e remover arquivos, mover um arquivo de um diretório ao outro, ou renomear um arquivo. O HDFS ainda não implementa cotas de usuários ou permissões de acesso. O HDFS não suporta *links* físicos ou *links* lógicos. Entretanto, a arquitetura do HDFS não impede a implementação dessas características.

O *NameNode* mantém o espaço nominal do sistema de arquivos. Qualquer mudança no espaço nominal do sistema de arquivos ou em suas propriedades é gravado no *NameNode*. Uma aplicação pode específicar o número de réplicas do arquivo que deveria ser mantido pelo HDFS. O número de cópias de um arquivo é chamado de fator de replicação daquele arquivo. Essa informação é armazenada pelo *NameNode*.

4.6 Replicação de Dados

O HDFS foi desenvolvido para armazenar com segurança arquivos muito grandes por meio de máquinas em um grande *cluster*. Ele armazena cada arquivo em uma sequência de blocos; todos os blocos de um arquivo, com a exceção do último bloco, são do mesmo tamanho. O tamanho dos blocos e o fator de replicação deles são configuráveis por arquivo. Uma aplicação pode especificar o número de réplicas de um arquivo. O mesmo fator de replicação pode ser especificado no momento da criação do arquivo e pode ser alterado mais tarde. Arquivos no HDFS são do tipo escreva-uma-vez e têm extritamente um escritor de cada vez.

O *NameNode* faz todas as decisões a respeito da replicação dos blocos. Ele periodicamente recebe um *Heartbeat* e um *Blockreport* de cada um dos *DataNodes* no *cluster*. O recebimento de um *Heartbeat* implica que o *DataNode* está funcionando corretamente. Um *Breakreport* contém uma lista de todos os blocos em um *DataNode*. A Figura 4.2 ilustra um conjunto de *DataNodes* que possuem blocos replicados de acordo com o fator de replicação descrito no arquivo de *Block Replication* no *NameNode*.

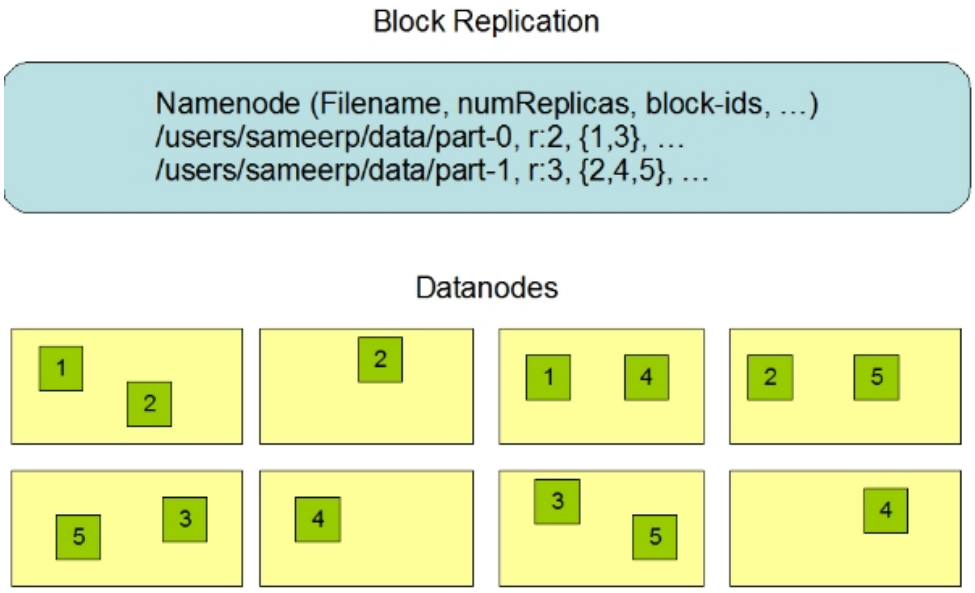

Figura 4.2: Replicação de Dados no HDFS [35].

4.6.1 Localização das Réplicas

A localização das réplicas é crítica para a performance e confiabilidade do HDFS. Otimizar a localização das réplicas distingue o HDFS da maioria dos outros sistemas de arquivos distribuídos. Essa é uma característica que necessita de muitos ajustes e experiencias. O propósito de uma política de localização de réplicas com conhecimento do *rack* utilizado é aumentar a confiabilidade, disponibilidade, e utilização da largura de banda da rede. A implementação atual para a política de localização das réplicas é o primeiro esforço nessa direção. Os objetivos de curto prazo de implementação dessa política são para validar isso no sistema de produção, conhecer mais sobre seu comportamento, e construir uma fundação para testar e pesquisar políticas mais sofisticadas.

Grandes instâncias do HDFS são executadas em um *cluster* de computadores que comumente se propagam através de muitos *racks*. A comunicação entre nós de diferentes *racks* deve ocorrer através de *switches*. Na maioria dos casos, a largura de banda da rede entre máquinas no mesmo *rack* é maior do que a largura de banda entre máquinas de diferentes *racks*.

O *NameNode* determina o *id* do *rack* que cada *DataNode* pertence por meio do processo descrito como conhecimento do *rack*. Uma simples, embora desotimizada, política é colocar as réplicas em *racks* únicos. Isso evita a perda de dados quando um *rack* inteiro falha e permite o uso da largura de banda por múltiplos *racks* quando se está lendo o arquivo. Essa política distribui uniformemente as réplicas do *cluster*, que torna fácil de se equilibrar a carga em caso de falha de componentes. Todavia, essa política aumenta o custo das escritas porque uma escrita necessita de transferir blocos para vários *racks*.

No caso comum, quando o fator de replicação é três, a política de localização do HDFS é colocar uma réplica em um nó do *rack* local, outra em um nó diferente do *rack* local, e a última em um nó diferente em um *rack* diferente. Essa política corta o tráfego de

escrita entre *racks*, que geralmente aumenta a performace de escrita. A chance de uma falha de *rack* é muito menor do que a de falha de um nó; essa política não impacta a garantia da confiabilidade e disponibilidade dos dados. Contudo, isso não reduz a largura de banda de rede utilizada na sua totalidade quando está lendo os dados, já que um bloco está localizado em apenas dois *racks* diferentes ao invés de três. Com essa política, as réplicas de um arquivo não se destribuem igualmente através dos *racks*. Um terço ds réplicas estão em um nó, dois terços das réplicas estão em um único *rack*, e mais um terço está distribuído uniformemente através dos *racks* restantes. Essa política aumenta a performance de escrita sem se comprometer com a confiabilidade dos dados e a performance de leitura.

A atual política de localização de réplicas padrão descrita ainda é um trabalho em progresso no HDFS [35].

4.6.2 Seleção das Réplicas

Para minimizar o consumo de largura de banda global e a latência de leitura, o HDFS tenta satisfazer uma requisição de leitura em uma réplica que está mais próxima do leitor. Se existe uma réplica no mesmo *rack* que o nó leitor, então aquela réplica é a que tem a preferência para satisfazer aquela requisição. Se um *cluster* HDFS compreende muitos *data centers*, então a réplica que é residente no *data center* local tem a preferência sobre qualquer réplica remota.

4.6.3 Modo Seguro

Ao inicializar, o *NameNode* entra em um estado especial chamado *Safemode* (modo seguro). A replicação de blocos de dados não ocorre enquanto o *NameNode* está no estado de modo seguro. O *NameNode* recebe uma mensagem *Heartbeat* e *Blockreport* dos *DataNodes*. Um *Blockreport* contém a lista dos blocos que um *DataNode* está hospedando. Cada bloco tem um número mínimo específico de réplicas. Um bloco é considerado replicado com segurança quando o número mínimo de réplicas daquele bloco de dados foi checado com o *NameNode*. Depois que uma porcentagem configurável de blocos de dados replicados com segurança checa com o *NameNode* (mais um adicional de 30 segundos, ou outro valor escolhido pelo usuário), o *NameNode* sai do estado de *Safemode*. Isso também determina a lista de todos os blocos de dados (se houver algum) que ainda tem menos do que o número especificado de réplicas. O *NameNode* estão replica esses blocos para outros *DataNodes*.

4.7 A Persistência dos Metadados do Sistema de Arquivos

O espaço nominal do HDFS é armazenado pelo *NameNode*. O *NameNode* usa um *log* de transação também chamado de *EditLog* para registrar persistentemente cada mudança que ocorre aos metadados do sistema de arquivos. Por exemplo, criar um novo arquivo no HDFS faz com que o *NameNode* insira um novo registro dentro do *EditLog* indicando isso. Similarmente, mudando o fator de replicação de um arquivo causa a inserção de um novo

registro dentro do *EditLog*. O *NameNode* usa um arquivo no seu sistema de arquivos local para armazenar o *EditLog*. O espaço nominal completo do sistema de arquivos, incluindo o mapeamento de blocos para arquivos e as propriedades do sistema de arquivos, estão armazenadas em um arquivo chamado *FsImage*. O *FsImage* é armazenado como um arquivo no sistema de arquivos local do *NameNode* também.

O *NameNode* mantém uma imagem de todo o espaço nominal do sistema de arquivos e arquiva na memória o *Blockmap*, que corresponde aos metadados associados a cada bloco no HDFS. Esse item metadado de chave é desenvolvido para ser compacto, tal que um *NameNode* com 4GB de RAM é suficiente para suportar um grande número de arquivos e diretórios. Quando o *NameNode* inicia, ele lê o *FsImage* e o *EditLog* do disco, aplica todas as transações do *EditLog* para a representação em memória do *FsImage*. Esse processso é chamado de *checkpoint*.

O *DataNode* armazena os dados do HDFS em seu sistema de arquivos local. O *DataNode* não tem conhecimento sobre os arquivos do HDFS. Ele armazena cada bloco de dados do HDFS em um arquivo separado no seu sistema de arquivos local. O *DataNode* não cria todos os arquivos no mesmo diretório. Ao invés disso, ele usa uma heurística para determinar o número ideal de arquivos por diretório e cria subdiretórios devidamente. Não é otimizado criar todos os dados locais no mesmo diretório porque o sistema de arquivos local pode não ser capaz de suportar eficientemente um grande número de arquivos em um mesmo diretório. Quando um *DataNode* é inicializado, ele examina por meio de seu sistema de arquivos local, gerando uma lista de todos os blocos de dados do HDFS que correspondem a cada um daqueles arquivos locais e envia o relatório para o *NameNode*, o qual é o *Blockreport* [35].

4.8 Robustez

O objetivo primário do HDFS é armazenar dados confiavelmente mesmo na presença de falhas. Os três tipos mais comuns de falhas são as falhas de *NameNode*, as falhas de *DataNode* e as partições na rede, as quais serão vistas nas próximas seções.

4.8.1 Falha do Disco de Dados, *Heartbeats* e Replicação

Cada *DataNode* envia uma mensagem de *Heartbeat* para o *NameNode* periodicamente. Uma quebra da rede pode causar a um subconjunto dos *DataNodes* a perda da conectividade com o *NameNode*. O *NameNode* detecta essa condição pela ausência de mensagens de *Heartbeat*. O *NameNode* marca os *DataNodes* sem *Heartbeats* recentes como mortos e depois disso não encaminham nenhuma requisição para eles. Qualquer dado que foi registrado para um *DataNode* morto não está disponível para o HDFS mais. A morte do *DataNode* pode causar a queda de alguns blocos para abaixo do seu valor especificado do fator de replicação. O *NameNode* constantemente busca quais blocos necessitam ser replicados e inicia a replicação quando necessário. A necessidade de replicação pode crescer devido a muitas rasões: Um *DataNode* pode vir a ficar indisponível, uma réplica pode ficar corrompida, um disco rígido de um *DataNode* pode ficar indisponível, ou o fator de replicação de um arquivo pode ser aumentado.

4.8.2 Balanceamento do *Cluster*

A arquitetura do HDFS é compatível com os esquemas de balanciamento de dados. Um esquema deve automaticamente mover os dados de um *DataNode* para outro, se o espaço livre no *DataNode* cair abaixo de uma certa marca. Na eventualidade de uma súbita alta demanda por um arquivo particular, um esquema pode dinamicamente criar réplicas adicionais e balancear outros dados no *cluster*.

4.8.3 Integridade dos Dados

É possível que um bloco de dados buscado de um *DataNode* esteja corrompido. Essa corrupção pode ocorrer por causa de falhas no dispositivo de armazenamento, falhas na rede, ou softwares defeituosos. O software cliente HDFS implementa uma verificação do tipo *checksum* no conteúdo dos arquivos do HDFS. Quando um cliente cria um arquivo no HDFS, ele computa um *checksum* de cada bloco do arquivo e guarda esses *checksums* em um arquivo escondido separado no mesmo espaço nominal no HDFS. Quando um cliente solicita o conteúdo de arquivos ele verifica se o dado recebido de cada *DataNode* corresponde com o *checksum* armazenado no arquivo de *checksum* associado. Se não, então o cliente pode optar por solicitar o bloco de outro *DataNode* que a tem uma réplica daquele bloco.

4.8.4 Falha no Disco de Metadados

O *FsImage* e o *EditLog* são as estruturas centrais do HDFS. A corrupção desses arquivos pode causar a instância do HDFS ficar sem funcionamento. Por essa razão, o NameNode pode ser configurado para suportar a manutenção de múltiplas cópias do *FsImage* e *EditLog*. Qualquer atualização tanto no *FsImage* quando no *EditLog* causam a cada um dos *FsImages* e *EditLogs* a serem atualizados síncronamente. Essa atualização síncrona de múltiplas cópias dos *FsImages* e *EditLogs* podem degradar a média de transações por segundo no espaço nominal que um *NameNode* pode suportar. Entretando, essa degradação é aceitável porque mesmo que as aplicações do HDFS sejam utilizadoras de dados intensivamente por natureza, elas não utilizam metadados intensamente. Quando um *NameNode* é reativado, ele seleciona o último consistente *FsImage* e *EditLog* para utilizar.

A máquina *NameNode* é um ponto de falha singular para um *cluster* HDFS. Se uma máquina *NameNode* falha, a intervenção manual se faz necessária.

4.9 Organização dos Dados

A organização dos dados no HDFS é feita por meio da sua estrutura de bloco de dados, dos estágios que constituem a organização dos dados e do *pipeline* de replicação do sistema de arquivos.

4.9.1 Blocos de Dados

O HDFS é formulado para suportar arquivos muito grandes. Aplicações que são compatíveis com o HDFS são aquelas que utilizam um grande conjunto de dados. Essas aplicações escrevem seus dados apenas uma vez, mas eles leem os dados uma ou mais vezes e requerem que essas leituras sejam satisfeitas em velocidade de fluxo. O HDFS suporta semânticas do tipo escreva-uma-vez-leia-várias nos arquivos. Um típico tamanho de bloco utilizado pelo HDFS é de 64MB. Então, um arquivo HDFS é colocado em pacotes de até 64MB, e se possível, cada pacote irá residir em um diferente *DataNode*.

4.9.2 Estágios

Uma requisição de um cliente para criar um arquivo não chega ao *NameNode* imediatamente. De fato, inicialmente, o cliente HDFS guarda os dados do arquivo em um arquivo local temporário. As escritas da aplicação são transparentemente redirecionadas para esse arquivo local temporário. Quando o arquivo local acumula dados acima do tamanho de um bloco do HDFS, o cliente contacta o *NameNode*. O *NameNode* insere o nome do arquivo dentro da hierarquia do sistema de arquivos e aloca um bloco de dados para ele. O *NameNode* responde à requisição do cliente com a identidade do *DataNode* e o bloco de dados de destino. Então o cliente envia o bloco de dados do arquivo local temporário para o especificado *DataNode*. Quando o arquivo é fechado, os dados remanescentes, ainda não enviados no arquivo local temporário, são transferidos para o *DataNode*. O cliente então avisa o *NameNode* que o arquivo está fechado. Neste ponto, o *NameNode* grava a operação de criação do arquivo em um armazenamento persistente. Se o *NameNode* cair antes do arquivo ser fechado, o arquivo será perdido.

A abordagem acima foi adotada depois de considerações cautelosas das aplicação alvo que são executadas no HDFS. Essas aplicações necessitam de escritas em fluxo para arquivos. Se um cliente escreve para um arquivo remoto diretamente sem nenhum *buffer* no lado do cliente, a velocidade de rede e o congestionamento da rede impactariam consideravelmente a vazão.

4.9.3 *Pipeline* de Replicação

Quando um cliente está escrevendo dados em um arquivo do HDFS, este dado é primeiramente escrito em um arquivo local, como explicado na última seção. Suponha que o arquivo do HDFS tenha um fator de replicação de três. Quando o arquivo local ocupar o tamanho de um bloco completo de dados do usuário, o cliente recebe uma lista de *DataNodes* do *NameNode*. A lista contém os *DataNodes* que irão hospedar uma réplica daquele bloco. O cliente então envia o bloco de dados para o primeiro *DataNode*. O primeiro *DataNode* inicia o recebimento dos dados em pequenas porções (4KB), escreve cada parte para o seu repositório local e transfere aquela parte para o segundo *DataNode* na lista. O segundo *DataNode*, por sua vez começa a receber cada parte do bloco de dados, escreve aquela para para o seu repositório e então envia aquela porção para o terceiro *DataNode*. Finalmente, o terceiro *DataNode* escreve os dados para seu repositório local. Então, um *DataNode* pode estar recebendo dados de um anterior no *pipeline* e ao mesmo tempo enviando dados para o próximo no *pipeline*. Então, os dados são colocados em um *pipeline* de um *DataNode* para o próximo.

4.10 Acessibilidade

O HDFS pode ser acessado por aplicações de muitas maneiras diferentes. Nativamente, o HDFS provê uma API Java para aplicações utilizarem. Um encapsulador da linguagem C para essa API Java também está disponível. Além disso, um *browser* HTTP pode também ser utilizado para navegar nos arquivos de uma instância HDFS.

4.10.1 FS *Shell*

O HDFS permite que os dados do usuário sejam organizados na forma de arquivos e diretórios. Ele provê uma interface de linha de comando chamada FS *shell* que permite que o usuário interaja com os dados no HDFS. A sintaxe desse conjunto de comandos é similar aos outros *shells* (e.g *bash*, csh) que os usuários já estão familiarizados. Aqui estão algumas amostras de pares ação/comando:

- chmod - Muda as permissões dos arquivos;
- cp - Copia arquivos de uma fonte a um destino;
- mv - Move o arquivo da fonte para o destino;
- rm - Remove arquivos especificados;
- rmr - Modo recursivo de exclusão.

O FS shell é direcionado para aplicações que necessitam que uma linguagem de script interaja com os dados armazenados.

4.10.2 DFSAdmin

O conjunto de comandos DFSAdmin é usado para administrar um *cluster* HDFS. Estes são comandos que podem ser usados somente por um administrador do HDFS. Aqui estão alguns exemplos de pares ação/comando:

- report - Informa estatísticas básicas do HDFS;
- safemode - É utilizado para entrada e saída do modo de segurança do HDFS;
- finalizeUpgrade - Remove *backups* prévios do *cluster* feitos durante o último *update*;
- refreshNodes - Atualiza o conjunto de *hosts* que podem se conectar ao *Namenode*.

4.10.3 Interface do Navegador

Uma instalação típica do HDFS configura um web *server* para expor o espaço nominal do HDFS através de uma porta TCP configurável. Isso permite que o usuário navegue no espaço nominal do HDFS e veja o conteúdo de seus arquivos utilizando um navegador web.

4.11 Requisição de Espaço

Um arquivo no HDFS possui políticas próprias também para ser deletado, permitindo que ocorra recuperação pós-falha e também mantendo o sistema balanceado após a diminuição do fator de replicação de algum arquivo.

4.11.1 Exclusão de Arquivos e Reinclusão

Quando um arquivo é deletado por um usuário ou aplicação, ele não é imediatamente removido do HDFS. Em vez disso, o HDFS, primeiramente, renomeia o arquivo para um arquivo no diretório *trash*. O arquivo pode ser restaurado rapidamente enquanto ele permanecer nesse diretório. Um arquivo permanece no diretório *trash* por uma quantidade de tempo configurável. Depois de expirar seu período de vida no diretório *trash*, o *NameNode* deleta o arquivo do espaço nominal do HDFS. A remoção de um arquivo faz com que os blocos associados com aquele arquivo estejam liberados. Note aqui que pode ter um tempo de atraso apreciável entre o tempo de uma exclusão de arquivo pelo usuário e o tempo do aumento correspondente de espaço livre no HDFS.

Um usuário pode reincluir um arquivo depois da exclusão enquanto o arquivo permanecer no diretório *trash*. Se um usuário quer reincluir o arquivo depois da exclusão, ele pode navegar para o diretório *trash* e recuperar o arquivo. O diretório *trash* contém apenas a última cópia do arquivo que foi excluído. O diretório *trash* é como qualquer outro diretório, com apenas uma característica especial, o HDFS aplica uma política específica para excluir arquivos automaticamente desse diretório.

4.11.2 Decréscimo do Fator de Replicação

Quando o fator de replicação de um arquivo é reduzido, o *NameNode* seleciona réplicas excedentes que podem ser deletadas. O próximo *Heartbeat* transfere esse informação para o *DataNode*. O *DataNode* então remove os blocos correspondentes e o correspondente espaço livre aparece no *cluster*. Mais uma vez, aqui pode ter um tempo de atraso entre a conclusão da chamada da API *setReplication* e o aparecimento do espaço livre no *cluster*.

Capítulo 5

Políticas de Replicação

Este capítulo discute o estudo de caso realizado na área de Bioinformática com base nas considerações teóricas feitas nos capítulos anteriores. Na Seção 5.1 são relatados diferentes modelos de armazenamento baseadas nas políticas de replicação. Na Seção 5.2 definiu-se os parâmetros do problema que deveriam guiar a implementação, e seriam primordiais para a concretização bem sucedida deste projeto. Na Seção 5.3, é feita a descrição do ambiente que foi utilizado durante a realização dos testes e experimentos no laboratório. Na Seção 5.4, apresenta-se quais foram as políticas utilizadas e faz considerações a respeito do uso de cada uma delas, relatando as dificuldades e as vantagens das diferentes abordagens. O estudo de caso é baseado na realização do processo de filtragem de arquivos do tipo FASTQ utilizados em *pipelines* de Bioinformática. Assim, são apresentadas as adaptações feitas no sistema de arquivos para melhor comportar a política de replicação proposta, que necessita da estruturação de blocos. Além disso, é apresentado o teste de tolerância a falhas e são feitas as comparações entre as políticas de replicação de dados utilizadas, apresentando o resultado obtido em cada uma delas. Ao final, é realizada uma comparação com o resultado obtido em uma pesquisa correlata para melhor compreensão das políticas estudadas.

5.1 Políticas de Replicação na Nuvem

As políticas de replicação são essenciais na computação em nuvem, pois são elas que permitem que haja disponibilidade escalar na nuvem. Mas, como se tratam de recursos que lidam diretamente com a maneira como os arquivos são utilizados, impactam a performance da execução, e por isso foram implementadas de diferentes maneiras buscando a maior eficiência no sistema de arquivos.

A política de replicação mais utilizada é a que replica os arquivos de maneira aleatória no sistema de arquivos. Nessa política, os arquivos são copiados de acordo com o fator de replicação e distribuídos dentro da nuvem. Nessa política, quanto maior o fator de replicação maior o número de computadores que possuem a réplica do arquivo.

Existe também a política de replicação baseada em blocos de dados. As políticas baseadas em blocos não copiam os arquivos completamente para um computador, mas antes dividem o arquivo em blocos, que podem ser definidos tanto pelo seu tamanho, ou seja, cada bloco terá um limite em bytes definido pelo programador; ou pode ser definido pela quantidade, com cada arquivo sendo dividido em um determinado número de blocos.

Cada um dos blocos do arquivo é enviado para um computador, o qual será o responsável por informar quais os blocos pertencentes ao arquivo que estão em sua posse.

A localidade representa como serão encontrados os arquivos na nuvem. E quanto mais próximos os arquivos estão do espaço onde serão utilizados, mais rápido é o acesso a eles, e por isso a localidade dentro da nuvem é de extrema importância ao definir a política de replicação.

5.2 Características da Implementação

Para melhor atender à demanda do processo de filtragem foram selecionadas características desse processo que deveriam fazer parte da implementação do sistema de computação a ser criado, de forma que contribuíssem para o aumento da performance. O levantamento das características foi feito com base na observação de como o processamento de dados biológicos ocorre. Dentre as características que deveriam fazer parte da implementação, estão:

- Arquivos de grande volume de dados no sistema de arquivos – os arquivos resultados do sequenciamento genético, geralmente, são do tamanho de *gigabytes*, podendo chegar sem dificuldades ao tamanho de *terabytes*. Isso ocorre devido à enorme complexidade das estruturas representadas dentro dos arquivos de bioinformática e à necessidade de alta qualidade e precisão dos arquivos usados.
- Acesso aos arquivos armazenados do tipo escreva-uma-vez-leia-várias – os arquivos depositados no sistema de arquivos da bioinformática raramente sofrem modificações, devido ao alto grau de precisão exigido para esses dados e são utilizados como base para muitos outros trabalhos. Tais características configuram um sistema de arquivos de pouca ou nenhuma alteração, mas muitas leituras e computações.
- Alta escalabilidade – a bioinformática é um ramo que tem crescido continua e rapidamente nos últimos anos, exigindo que o desenvolvimento de qualquer sistema para bioinformática esteja apto a escalar horizontalmente sem dificuldades.
- Processos de alto custo de memória principal – o processamento de grandes arquivos está constantemente relacionado aos altos custos de memória principal também, tanto devido à necessidade do armazenamento de trechos do arquivo diretamente em memória, quanto devido à necessidade de armazenar resultados parciais do processamento para operações posteriores.
- Computação intensa e de longa duração – a computação de arquivos da ordem de *terabytes* pode levar horas, ou até mesmo dias dependo da sua finalidade, com isso se faz necessária a utilização de softwares que suportem a ocorrência de falhas e um conjunto de hardware de alta capacidade de processamento.
- Paralelismo e multiprocessamento – para realizar a execução do processo de maneira mais rápida deve-se utilizar dos recursos de multiprocessamento, por meio dos diversos núcleos dos processadores presentes nas máquinas, e do paralelismo, habilidade de alocar diversas máquinas para a resolução de um mesmo problema.

5.3 Ambiente Utilizado

Para desenvolver a nuvem computacional descrita foi utilizado o Apache Hadoop, devido à sua eficiência no gerenciamento de arquivos e à possibilidade de se configurar nossa própria política de replicação dentro do sistema de arquivos. Além disso, o *Hadoop Distributed File System* (HDFS) permite que computadores comuns façam parte do *cluster*, possibilitando a computação com máquinas heterogêneas de baixo custo.

Para realizar o paralelismo nos processos de computação escolheu-se o utilitário *streaming* do Hadoop [6], que permite ao usuário criar suas rotinas dentro do sistema de arquivos do Hadoop e executar o programa criado como um *job* do próprio Hadoop. Com a utilização do *streaming* o usuário pode definir pastas e/ou arquivos que irá utilizar durante a execução e também define o diretório para o qual os arquivos resultantes da computação serão enviados.

Quatro máquinas foram utilizadas para realizar o estudo de caso, mas o propósito deste estudo é que seja aplicável para qualquer número de computadores. A configuração dos computadores era:

Primeira Máquina:

- 1 Processador Intel Xeon E3-1220 com 8 núcleos de 3.1-3.4 GhZ de frequência;
- 12 gigabytes de memória RAM;
- 1 terabyte de armazenamento;
- Sistema operacional Linux na distribuição Ubuntu 10.04.

Segunda Máquina:

- 1 Processador Intel Xeon E3-1220 com 8 núcleos de 3.1-3.4 GhZ de frequência;
- 8 gigabytes de memória RAM;
- 1 terabyte de armazenamento;
- Sistema operacional Linux na distribuição Ubuntu 12.04.

Terceira Máquina:

- 1 Processador Intel Core i7-2630QM com 4 núcleos de 2.0-2.9 GhZ de frequência;
- 4 gigabytes de memória RAM;
- 500 gigabytes de armazenamento;
- Sistema operacional Linux na distribuição Ubuntu 12.04.

Quarta Máquina:

- 1 Processador Intel Dual Core T2370 com 2 núcleos de 1.76 GhZ de frequência;
- 2 gigabytes de memória RAM;

- 120 gigabytes de armazenamento;
- Sistema operacional Linux na distribuição Ubuntu 12.04.

Todas as máquinas utilizaram o Apache Hadoop na versão 0.20 para o armazenamento dos dados na nuvem. A Figura 5.1 ilustra a nuvem computacional utilizada.

Figura 5.1: Nuvem Computacional com as Quatro Máquinas Utilizadas.

Para testar a capacidade de cada um dos computadores utilizados foi feito uma *benchmark* com um arquivo FASTQ para filtragem durante a pequisa. Os tempos de duração dos processos foram gravados para cada um dos computadores, como pode ser visto na Tabela 5.1, e dessa maneira foi possível conhecer mais precisamente o potencial dos recursos de cada uma das máquinas para a execução do processo de filtragem. A partir de então, foi necessário apenas descobrir a porcentagem de processamento que cada máquina estava utilizando para que fosse possível determinar quanto da capacidade de processamento estava disponível, e assim a computação pode ser distribuída de forma igualitária.

Tabela 5.1: Tempo Total de Execução em Cada uma das Máquinas.

Máquina	Tempo Utilizado (Segundos)
1	217.3
2	214.6
3	335.8
4	1322.5

5.4 Estudo de Caso

Primeiramente realizou-se um teste base, sem políticas de replicação, para verificar a qualidade de uma execução simples em face das exigências do processo de filtragem. Vale ressaltar que o processamento dos arquivos com os dados vindos do sequenciador ainda ocorre em alguns laboratórios de Bioinformática.

Para a execução do teste utilizou-se um arquivo FASTQ com mais de três *gigabytes* de tamanho, além disso adicionou-se nos comandos de filtragem a realização de gráficos, que é uma das tarefas de maior processamento para esse tipo de arquivo, e permite que as análises das sequências sejam feitas de maneira mais simples, pois ao invés de colocar as sequências e qualidades separadas em texto, os gráficos mostram as sequências junto com as suas respectivas qualidades, de forma compacta e intuitiva, além de mostrar estatísticas e informações não triviais que muito dificilmente seriam observadas dentro um arquivo tão grande.

As principais dificuldades na falta de políticas de replicações que puderam ser observadas são a falta de segurança no caso de falhas, a ausência de um esquema que possibilite que computadores possam auxiliar na computação e a necessidade de um supercomputador para que seja possível completar o processamento sem exceder a capacidade da memória principal, embora conte com a vantagem de não reduzir sua capacidade na proporção do fator de replicação.

Posteriormente, utilizou-se a política de replicação de dados convencional para realizar o processo de filtragem. A política convencional replica os dados dentro do sistemas de arquivo de forma aleatória, de acordo com o fator de replicação estabelecido.

O principal problema dessa política de replicação encontra-se na dificuldade de atender as necessidade de paralelismo, pois para que todos os computadores pertencentes a essa nuvem computacional pudessem colaborar na execução dos arquivos, apenas duas soluções estariam disponíveis. Na primeira seria necessário que todo o sistema de arquivos estivesse armazenado em cada um dos computadores, o que é naturalmente inviável em *clusters* com muitos computadores. Na segunda teríamos de aceitar o fato de manter o número de réplicas e a cada requisição enviar o arquivo para os nós participantes do processamento. O problema associado a essa segunda solução reside no fato de que como se tratam de arquivos da ordem de *gigas* e *terabytes*, o tempo de leitura do disco, de transmissão dos arquivos e o tráfego na rede se tornam tão altos que não compensam a utilização de outras máquinas no processamento.

A partir de então passou-se a testar as replicações que particionam os dados antes de colocá-los no sistema de arquivos, ao invés de colocá-los completamente no sistema de arquivos de uma única vez. O HDFS utiliza a replicação por blocos, ou seja, cada bloco no HDFS é replicado para outros computadores, mas a política de replicação de dados do HDFS prioriza o balanceamento, e com isso distribui randomicamente as réplicas dos blocos de dados dentro do sistema de arquivos, levando a configurações de localidades que não contribuem tanto com a computação. A Figura 5.2 é um exemplo de como a distribuição de dados aleatória pode interferir na fase de computação, no caso, as réplicas foram distribuídas aleatoriamente entre os *datanodes*, que estão representados por retângulos de cor creme, com o fator de replicação três. O arquivo foi divididos em quatro blocos, e mesmo utilizando da estrutura de blocos para distribuir melhor as réplicas, o fato de distribuir aleatoriamente não garante que todos os nós terão réplicas para executar a computação.

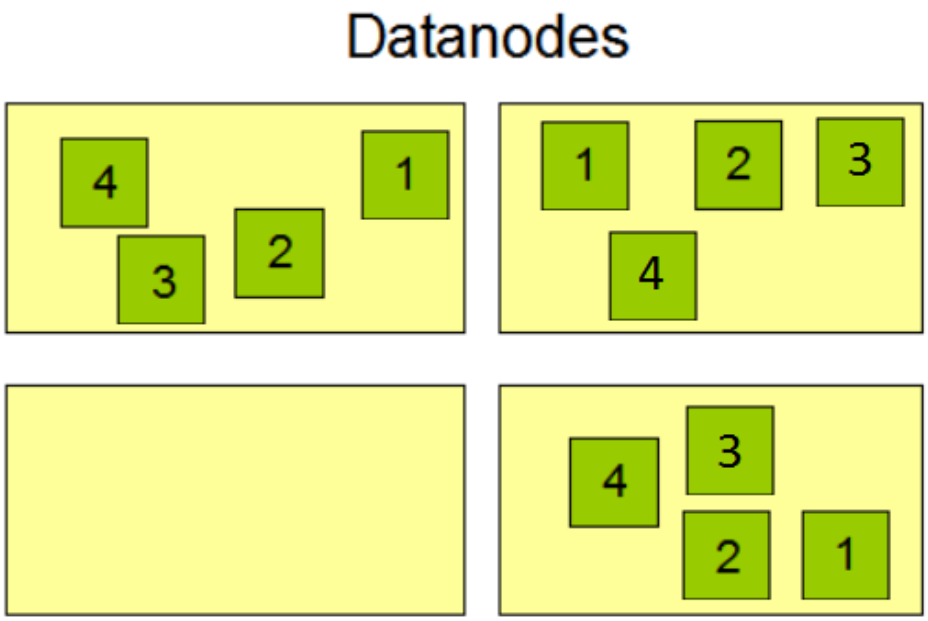

Figura 5.2: Exemplo de Distribuição de Blocos Aleatória.

Para resolver o problema da aleatoriedade é necessário utilizar API's que permitam ao usuário do HDFS desenvolver a sua própria política de replicação para os blocos. O HDFS-385 [7] é uma API que executa a rotina de controle de localidade dentro do HDFS e deixa a cargo do programador escolher qual algoritmo irá definir a localização das réplicas dentro do HDFS. A possibilidade de definição customizada é importante, já que além de tornar o sistema mais configurável (atendendo melhor às necessidades de cada usuário), viabiliza o teste de diferentes políticas no HDFS.

Como resultado da utilização de políticas de replicação próprias, tem-se uma configuração de arquivos com uma distribuição muito melhor, permitindo que todos os computadores que foram adicionados para a replicação pudessem participar da fase de computação. Na Figura 5.3 é possível notar a diferença de distribuição entre a política de replicação proposta, com blocos localizados por meio de um algoritmo de balanceamento, e a política aleatória da Figura 5.2.

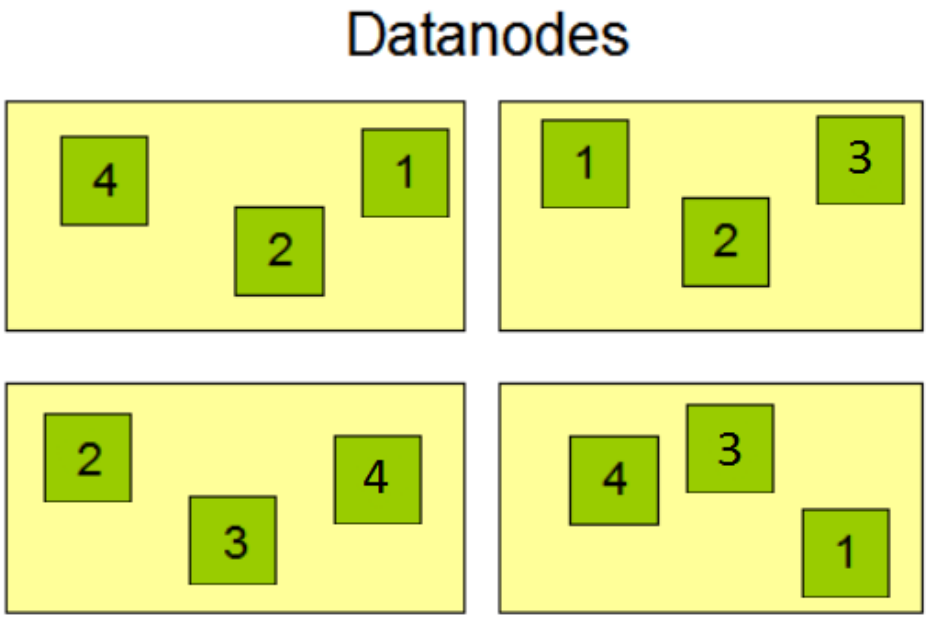

Figura 5.3: Exemplo de Distribuição de Blocos Com Política de Replicação Definida.

Entretanto, utilizar a política de replicação apenas não é suficiente para que se possa desfrutar de seus benefícios. O processo de filtragem requer que os dados contidos no arquivo FASTQ estejam em determinada ordem, e essa ordem deve ser respeitada durante a replicação para evitar que parte dos dados fique sem ser filtrada. Segue um exemplo da organização de um arquivo FASTQ na Figura 5.4.

```
@seq3 length=100
TAGATTGCGTGCGTACGTGTGTGCATGCGTTGTGCCGCGCTCTGTGCGTGTACGTGACGTGACGTGGTGTATAGATTGCGTGCGTACGTGTGTGCATGCG
+seq3 length=100
FHHFFFFFFDDAA@====AAB===BBBBAAADDDDDDDDAAAADDDDD?????FFFFF??FFFFFFDA@AFFFFFFFFFFFFFFFFCCABA?>9:773..
@seq3_dupl1 length=100
TAGATTGCGTGCGTACGTGTGTGCATGCGTTGTGCCGCGCTCTGTGCGTGTACGTGACGTGACGTGGTGTATAGATTGCGTGCGTACGTGTGTGCATGCG
+seq3_dupl1 length=100
FHHFFFFFFDDAA@====AAB===BBBBAAADDDDDDDDAAAADDDDD?????FFFFF??FFFFFFDA@AFFFFFFFFFFFFFFFFCCABA?>9:773..
@seq4 length=50
TAGATTGCGTGCGTACGTGTGTGCATGCGTTGTGCCGCGCTCTGTAGAGA
+seq4 length=50
???CCDDBBBBBAA333:ABB=:::AGFFFFFHHHHFFFFFFFF??@FFF
@seq5 length=100 Ns_begin=10
NNNNNNNNNNTACACCAGAGGTGTCTCTGTGTGGGGCCTGTGTGCCAAAAGTGAGAGTTGAGAAGAGGCGTGGAGGAGATGACACACCCCGTGTGTTCTC
+seq5 length=100 Ns_begin=10
```

Figura 5.4: Exemplo no Formato FASTQ.

5.4.1 Adaptações no Sistema de Arquivos

Para respeitar os critérios de organização do arquivo FASTQ na realização da filtragem, e enfim poder utilizar a política de replicação corretamente, foi necessário realizar algumas adaptações no sistema de arquivos que pudessem suprir essa necessidade, essas adaptações foram divididas em quatro fases, as quais foram:

- **Fase 1** – em primeiro lugar, os blocos de dados no sistema de arquivos não foram colocados diretamente no HDFS, já que caso os arquivos tivessem sido armazenados diretamente não seria possível prever quando ocorreria o término do bloco de dados, podendo resultar na quebra de sequências de filtragem. Em vez disso, os blocos passavam por uma rotina de tratamento antes da inserção no banco de dados, essa rotina determinava até qual parte do arquivo seria inserida em cada bloco, e dessa maneira tem-se a certeza de que todos os blocos são consistentes para o processo de filtragem. As etapas da rotina de inserção podem ser melhor compreendidas analisando a Figura 5.5

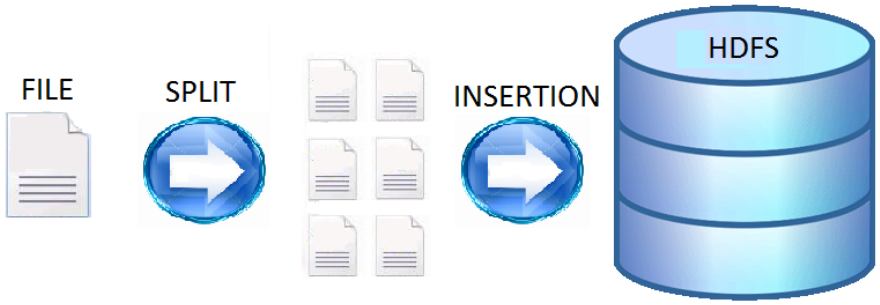

Figura 5.5: Etapas da Rotina de Inserção.

- **Fase 2** – como cada parte do arquivo se tornou um bloco distinto, deve-se conhecer aonde cada um dos blocos foi alocado, e dessa forma escolher os computadores que participarão da fase de processamento do arquivo. Com esse intuito é realizado um mapeamento dos computadores que possuem o arquivo durante a fase de criação das réplicas, esse mapeamento constitui simplesmente de um arquivo contendo o identificador dos blocos e o local onde cada um foi replicado. Uma vez realizada a identificação, é iniciada a execução dos *jobs* referentes aos blocos por meio do *streaming*.
- **Fase 3** – em terceiro lugar, foi utilizado um programa Java para alocação de recursos e gerenciamento da tarefa, pois mesmo o Hadoop possuindo um sistema de arquivos multiplataforma e com a capacidade de determinar o número de tarefas que podem ser executadas em um mesmo *datanode*, há uma dificuldade em se conhecer o estado corrente do sistema que está executando o processamento. Por exemplo, um computador que possui quatro núcleos pode estar fixado para executar no máximo oito tarefas ao mesmo tempo, por padrão, mas dependendo da atividade agendada dentro da nuvem, alguma tarefa criada pode estar utilizando vários núcleos do processador e o processamento das duas pode ser prejudicado. Da mesma forma, podem estar sendo executadas oito tarefas que não demandam toda a capacidade de processamento do computador e chegar uma tarefa que poderia utilizar os recursos disponíveis sem prejuízo algum aos outros processos, mas essa tarefa terá de aguardar na fila de execução.

- **Fase 4** – em último lugar fica a fase de alocação dos novos arquivos gerados após o processamento. Essa fase realiza a primeira etapa novamente para os arquivos criados, gera os *logs* do processamento e efetua o mapeamento descrito na fase dois.

5.4.2 Teste de Tolerância a Falhas

Para comparar as políticas de replicação mediante a falha fez-se um teste de queda do serviço durante a execução do programa, através da simples retirada do fio que ligava o computador à rede de testes. Primeiramente, quando um computador de um sistema que não utiliza política de replicação fica indisponível todo o serviço fica indisponível enquanto o computador não voltar a ser identificado, já no caso de falha em um computador de um sistema que possui a política de replicação convencional, o serviço continuava a ser executado, porém a velocidade de execução caia significativamente, pois os computadores que possuem alguma dentre as poucas réplicas existentes são os principais responsáveis pela performance da execução.

Observando a política de replicação com réplicas no nível de blocos, foi possível notar uma melhora significativa na resposta em caso de falhas, visto que todos os blocos estavam replicados nos computadores que participariam da execução, e assim não houve queda de rendimento além daquela correspondente à razão entre a capacidade de computação daquele computador em relação à soma da capacidade dos demais, como pode-se verificar na Figura 5.6.

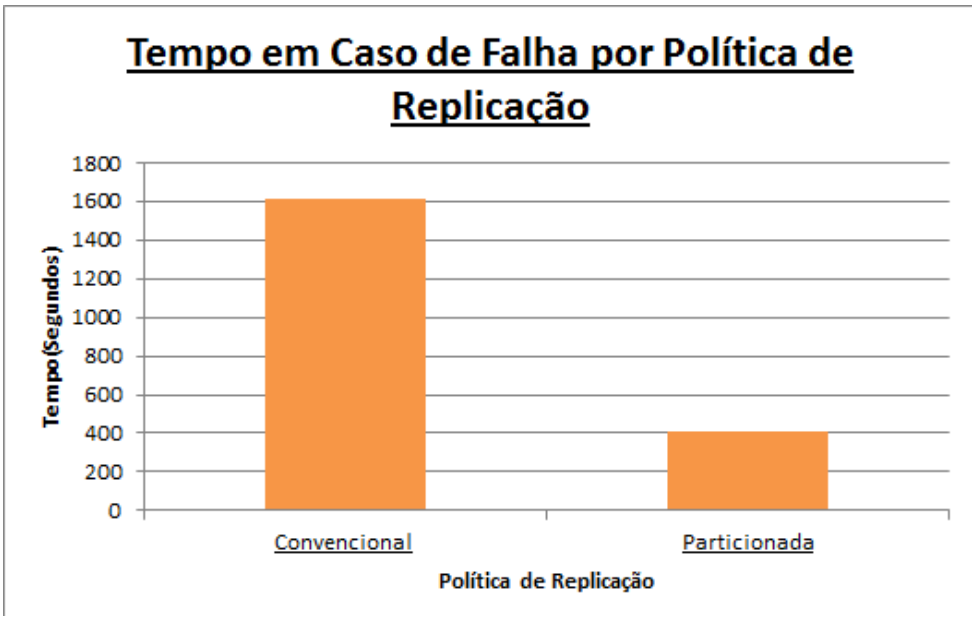

Figura 5.6: Tempo de Utilizado por Cada Política de Replicação em Caso de Falha.

5.4.3 Resultados e Comparações entre as Políticas de Replicação

Primeiramente, será analisada a política de replicação que usou os arquivos completos para a replicação no sistema de arquivos. Os testes realizados apresentaram os resultados apontados pela Figura 5.7. Na medida que o fator de replicação cresce, o número de computadores que participam do processamento também aumenta. Pode-se observar na

Figura a gradativa melhora no tempo que é proporcional ao crescimento do fator de replicação e à capacidade dos computadores que passam a participar da fase de computação.

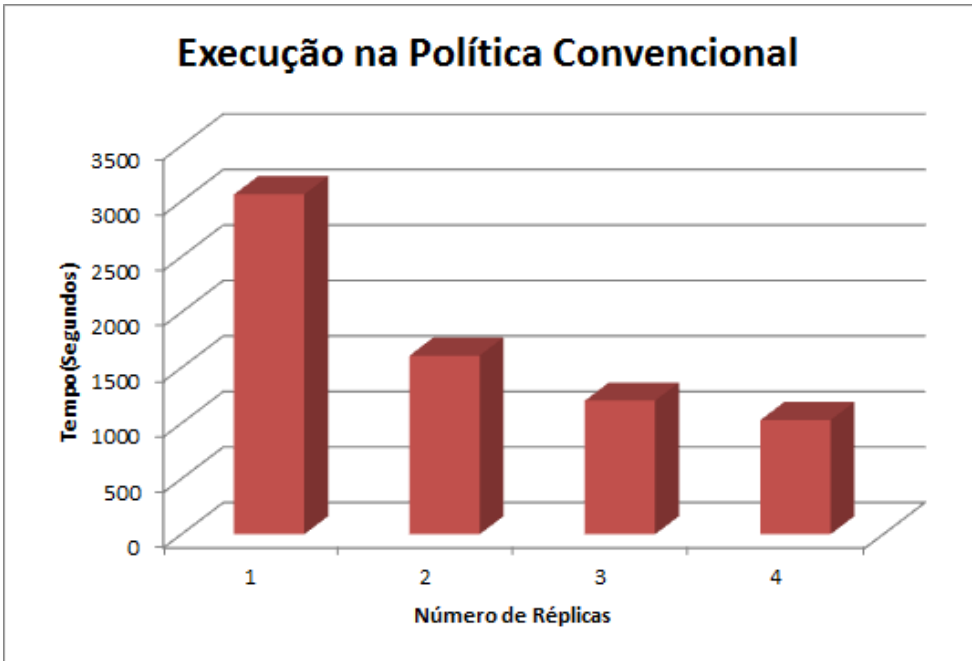

Figura 5.7: Tempo de Filtragem por Fator de Replicação.

Resta então fazer a análise dos resultados da política de replicação proposta nessa pesquisa, que dividiu os arquivos em blocos antes de inserir no sistema de arquivos. A Figura 5.8 corresponde aos resultados obtidos através da utilização dessa política de replicação. Os testes foram realizados com o fator de replicação 3, todavia os mesmos resultados seriam obtidos se os dados fossem replicados 1 ou mais vezes, pois todas as máquinas já participavam da fase de computação desde a primeira replicação no sistema de arquivos, em virtude do fato de 32 blocos serem mais do que suficiente para suprir todas as 4 máquinas com pelo menos um bloco. Entretanto, o valor três foi utilizado para simular a necessidade de recuperação a falhas vista na subseção 5.4.2. Na Figura 5.8, cada bloco é executado por uma unidade de processamento, por isso, o desempenho cresce proporcionalmente ao número de blocos no sistema de arquivos.

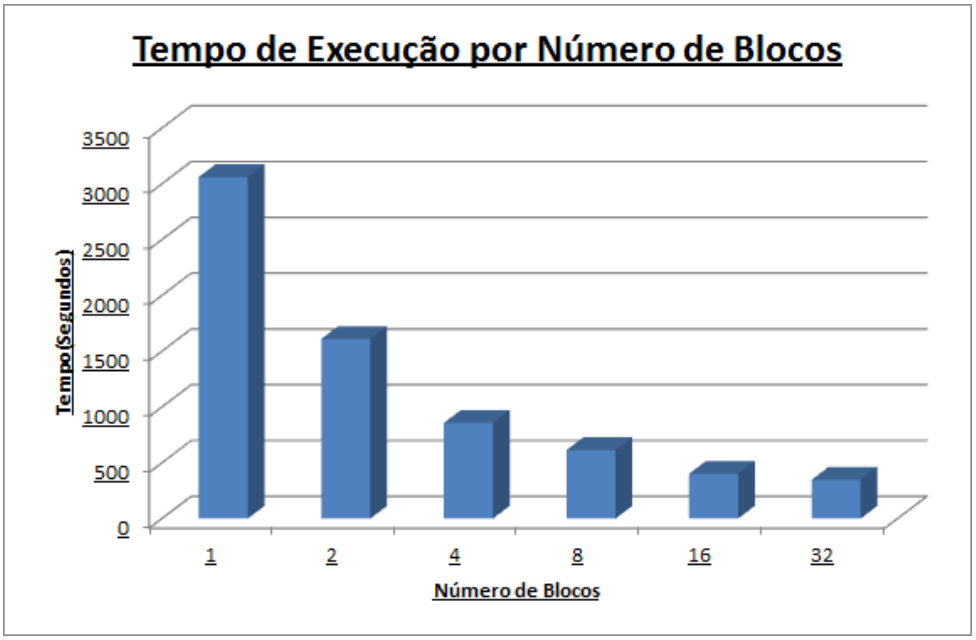

Figura 5.8: Tempo de Filtragem por Número de Blocos.

Pode-se analisar na Figura 5.8 outro fator de grande relevância para os resultados, apesar do sistema contar apenas com 14 núcleos de processamento, ao aumentar o número de blocos do arquivo de 16 para 32 houve uma melhora no desempenho. Isso se deve, principalmente, ao fato de que os computadores mais modernos utilizam *threads* e uma série tecnologias para aumentar a performance de seus processadores quando eles estão em alta taxa de atividade. Além disso, a filtragem é um processo que provoca muitas interrupções no barramento, pelo fato de ter que ler uma quantidade tão grande de dados em disco, isso faz com que o processador fique ocioso no caso de não ter outros processos para utilizar, algo que não ocorre com tanta facilidade quando há mais processos do que núcleos de computação.

A Figura 5.9 efetua a comparação entre os tempos de computação utilizados pelas duas políticas de replicação levando em conta um fator de replicação padrão definido como três. Pelos resultados obtidos pode-se observar a superioridade em qualidade temporal da política de replicação em blocos, que permite que mais computadores participem do processamento e ainda possibilita que todos os núcleos dos computadores participem ativamente da execução.

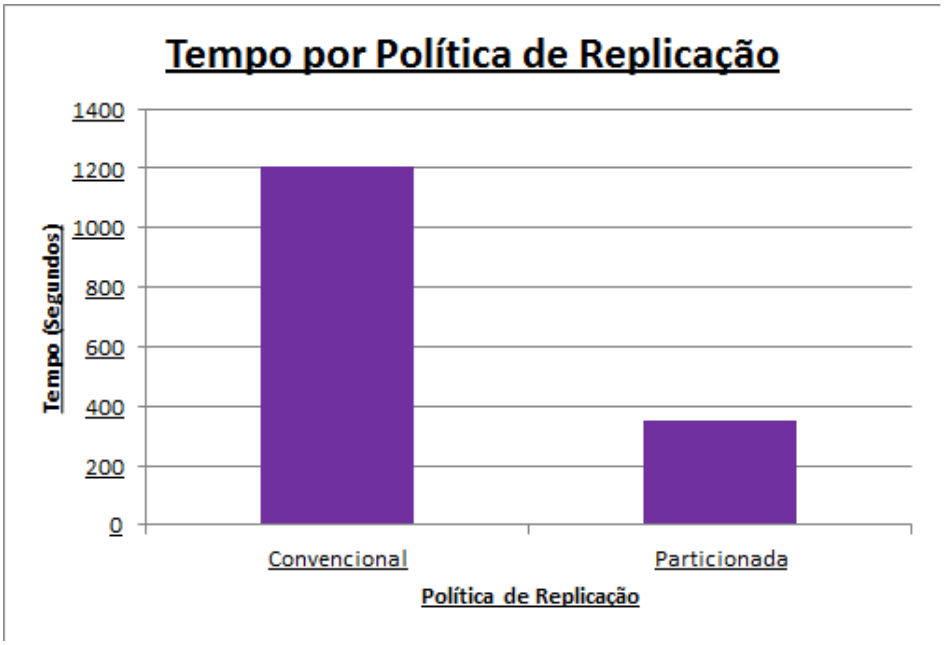

Figura 5.9: Tempo Utilizadação por Cada Política de Replicação, Usando-se o Fator de Replicação 3.

5.5 Comparações com Trabalho Correlato

Estudos de bioinformática e de replicação de dados já foram realizados em diversos trabalhos, e para mostrar como a maneira de replicar os dados influencia nos resultados devido às suas diferentes características, realizou-se uma comparação entre as políticas adotada em outra pesquisa e a política proposta nesTe trabalho.

Em [20], é feita a filtragem de arquivos FASTQ durante o estudo de caso realizado. Na pesquisa realizada não foi utilizada nenhuma política de replicação, e a filtragem foi realizada sem a utilização de componentes gráficos levou cerca de 2 minutos e 16,160 segundos para ser concluída.

A filtragem foi feita no computador denominado "serina", pertencente ao laboratório de Bioinformática da UnB, que possui 24 *gigabytes* de memória RAM, e utiliza dois processadores Intel®Core™2 Quad 2.4 GhZ, num total de 8 núcleos de processamento. Para ser efetuada uma comparação utilizando a política de replicação proposta neste trabalho realizou-se um teste de filtragem com as mesmas características, utilizando apenas as máquinas 1 e 3, descritas na seção 5.3, que somadas possuem oito núcleos, a mesma quantidade de núcleos da serina, e o processo de filtragem foi concluído no tempo de 37.92 segundos.

Pode-se notar claramente que a filtragem foi realizada de maneira muito mais rápida utilizando a política de replicação sugerida, como pode-se observar no gráfico comparativo impresso na Figura 5.10 e isso ocorre, mais uma vez, devido ao uso otimizado dos computadores e seus respectivos núcleos de processamento, proporcionados pela política de replicação de dados sugerida.

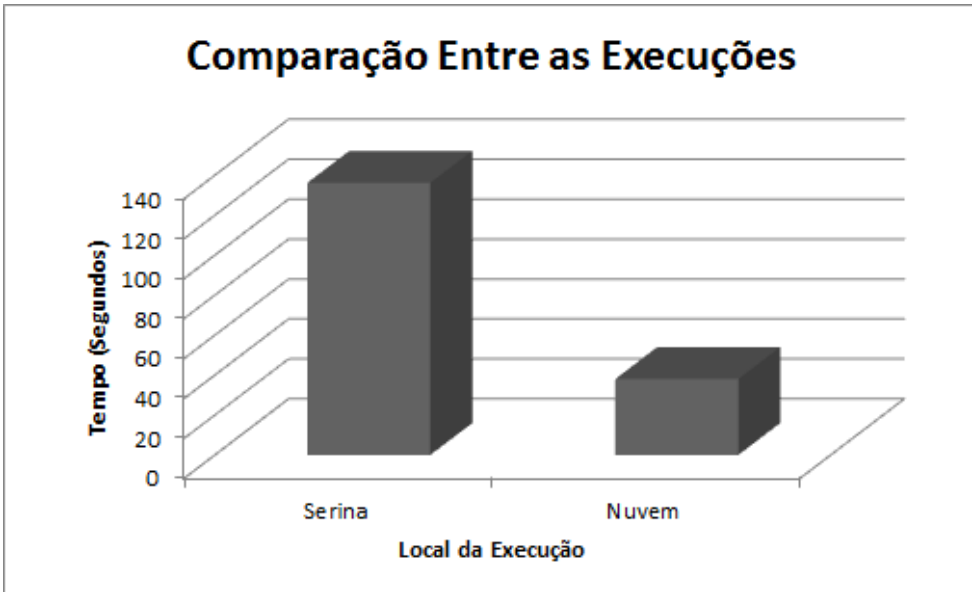

Figura 5.10: Comparação entre os Tempos de Filtragem.

Capítulo 6

Conclusão e Trabalhos Futuros

Atualmente, muitos laboratórios de bioinformática utilizam supercomputadores para realizar o processamento dos arquivos de filtragem, para evitar os problemas relacionados ao fato da computação frequentemente exceder a capacidade de memória de computadores comuns e devido ao alto poder de computação necessário para realizar as tarefas provenientes dessa atividade. Entretanto, com a utilização da computação em nuvem é possível a obtenção de recursos de maneira rápida e simples para atender as necessidades do processo de computação. As políticas de replicação tornam ainda mais ágeis o processamento distribuído que é realizado na nuvem, aumentando a disponibilidade dos arquivos sem necessariamente aumentar seu fator de replicação. Assim, é possível administrar os recursos computacionais de forma adequada eficiente e não se requer necessariamente que o usuário disponha de supercomputadores para realizar a computação, pois ao invés de realizar todo o processamento de uma única vez, o processamento é dividido em várias etapas de forma que o sistema de computação resultante permita que o processo seja executado na mesma velocidade, mas com estágios reduzidos que não exigem tanta memória.

Pode-se observar também que a utilização da política de replicação correta permite que haja o aproveitamento total dos recursos computacionais disponíveis, tanto em capacidade de processamento, quanto de utilização de memória RAM e de velocidade de leitura e capacidade de armazenamento. Não atuando apenas na gestão de recursos computacionais, com a utilização de políticas de replicação é possível criar uma camada de pré-processamento seletivo, localizando de forma eficiente as réplicas criadas, criando réplicas distribuídas inteligente e administrando como devem ser utilizadas cada uma delas.

Destarte, é possível averiguar a alta escalabilidade que a política de replicação de dados por blocos localizados pode trazer, visto que, mesmo mantendo um fator de replicação baixo, quando é requisitada a computação, todos os computadores presentes na nuvem podem participar do processamento, utilizando o máximo de seus recursos computacionais específicos. E pode-se constatar, a partir dos dados apresentados, que a replicação de dados permite que a computação dentro do processo de filtragem ocorra de maneira mais segura, reduzindo as perdas no caso de falhas, evitando o retrabalho e aumentando a disponibilidade no acesso aos dados do sistema de arquivos.

Como trabalhos futuros, primeiramente, é necessário exportar esse ambiente para um SGBD NoSQL, para que se possa utilizar uma organização mais simples do sistema de arquivos numa camada superior. Além disso, a utilização de um SGBD NoSQL permitiria que os serviços fossem disponibilizados para fora da nuvem com maior facilidade,

podendo até haver uma computação paralela entre nuvens computacionais baseada na mesma política de replicação utilizada.

Além dessas alterações, é importante também utilizar o hardware adequado para as operações, e durante a pesquisa muitas questões foram levantadas a respeito da utilização das GPUs (*Graphic Processing Units*), pois é um dos recursos presente nos computadores utilizados e não teve participação alguma no estágio de computação. Pesquisas recentes têm apontado as GPUs como unidades de processamento excepcional para o tipo de tarefa realizada e utilizar esse recurso pode fazer grande diferença nos resultados apresentados.

Referências

[1] 10gen. The MongoDB Company. http://www.10gen.com/. Acessado em 20-06-2013. 14

[2] D. J. Abadi. Data management in the cloud: Limitations and opportunities. *Data Eng. Bull*, 32:3–12, 2009. 2

[3] Amazon. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/pt/ec2/. Acessado em 15-07-2013. 8

[4] Amazon. Descrição Detalhada. http://aws.amazon.com/pt/dynamodb/#details. Acessado em 22-06-2013. vii, 18

[5] Apache. Architecture Overview. http://hbase.apache.org/book/architecture.html#arch.overview. Acessado em 22-06-2013. vii, 12, 14

[6] Apache. Hadoop Streaming. http://hadoop.apache.org/docs/stable/streaming.html. Acessado em 22-06-2013. 34

[7] Apache. Hdfs-385. https://issues.apache.org/jira/browse/HDFS-385. Acessado em 22-06-2013. 37

[8] Apache. Welcome to Apache Cassandra. http://cassandra.apache.org. Acessado em 20-06-2013. 10

[9] Apache. Welcome to Apache Zookeeper. http://zookeeper.apache.org. Acessado em 22-06-2013. 13

[10] A. Bento and R. Bento. Cloud computing: It change or management revolution? *Economics Conference Program*, pages 20–52, 2010. 4

[11] E. A. Brewer. Towards robust distributed systems. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, PODC '00, pages 7+, New York, NY, USA, 2000. ACM. vii, 1, 4

[12] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, June 2009. 2

[13] E. Ciurana. Developing with google app engine, 2009. Apress, Berkely, CA, USA. 8

[14] D. Cutting. História do Apache Hadoop. http://labs.yahoo.com/files/cutting.pdf. Acessado em 28-01-2013. 20

[15] Embarcadero. Database trends survey report. http://www.embarcadero.com/images/dm/technical-papers/database-survey-report.pdf, 2010. 2

[16] Eucalyptus. Why Eucalyptus. http://www.eucalyptus.com/why-eucalyptus. Acessado em 15-07-2013. 8

[17] Google. Docs. http://docs.google.com. Acessado em 15-07-2013. 7

[18] Google. MapReduce: Simplified Data Processing on Large Clusters. http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/pt-BR//archive/mapreduce-osdi04.pdf. Acessado em 22-06-2013. 20

[19] Google. The Google File System. http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/pt-BR//archive/gfs-sosp2003.pdf. Acessado em 22-06-2013. 20

[20] R.C. Huacarpuma. Modelo de Dados para um Pipeline de Sequenciamento de Alto Desempenho Transcritômico, March 2012. 43

[21] J. C. S. Junior. Ácidos Nucléicos. http://www.professorjarbasbio.com.br/pagina9.htm. Acessado em 28-01-2013. 21

[22] H. Lamehamedi and outros. Data replication strategies in grid environments. *Algorithms and Architectures for Parallel Processing*, 5:378–383, 2002. 2

[23] P. Mell and T. Grance. The NIST Denition of Cloud Computing. http://csrc.nist.gov/groups/SNS/cloud-computing/cloud-def-v15.doc, 2009. vii, 5, 7, 8

[24] Microsoft. Windows azure. http://www.windowsazure.com/pt-br/?l=pt-br. Acessado em 15-12-2012. 8

[25] MongoDB. Agile and Scalable. http://www.mongodb.org. Acessado em 20-06-2013. 14

[26] MongoDB. Replica set fundamental concepts. http://docs.mongodb.org/manual/core/replication/. Acessado em 28-01-2013. vii, 14, 15

[27] M. C. Nogueira and D. C. Pezzi. A computação agora é nas nuvens, 2009. Technical Report - Universidade de Cruz Alta, Cruz Alta, Rio Grande do Sul, Brasil. 2

[28] S. Perera. Considerações sobre o banco de dados apache cassandra. http://www.ibm.com/developerworks/br/library/os-apache-cassandra/. Acessado em 28-01-2013. 11

[29] Salesforce. Costumer Relationship Management. http://www.salesforce.com/crm/. Acessado em 15-07-2013. 7

[30] G. Shapira. Replication is the new durability (thoughts about dynamo). http://www.pythian.com/blog/replication-is-the-new-durability-thoughts-about-dynamo. Acessado em 28-01-2013. 17

[31] F. R. C. Sousa, L. O. Moreira, J. A. F. de Macêdo, and J. C. Machado. Gerenciamento de dados em nuvem: Conceitos, sistemas e desafios. *Tópicos em Sistemas Colaborativos, Interativos, Multimídia, Web e Bancos de Dados*, 1:101–130, 2010. 2

[32] D. Thakur. Technical Report – What is Data Replication? Advantages & Disadvantages of Data Replication. http://ecomputernotes.com/database-system/adv-database/data-replication. Acessado em 22-06-2013. 2

[33] M. Vardanyan. Os bancos de dados NoSQL, uma olhada no HBase. http://imasters.com.br/artigo/22216. Acessado em 28-01-2013. 13

[34] W. Vogels. Amazon Dynamo. http://www.allthingsdistributed.com/2007/10. Acessado em 28-01-2013. 17

[35] Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media, original edition, June 2009. vii, 20, 22, 25, 26, 27