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

Uma Solução de Checkpoint em Aplicações de Bioinformática para a Plataforma de Nuvem Federada BioNimbuZ

Daniel Ferreira Schulz

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

Orientador
Prof.a Dr.a Aletéia Patrícia Favacho de Araújo

Brasília
2018

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

Uma Solução de Checkpoint em Aplicações de Bioinformática para a Plataforma de Nuvem Federada BioNimbuZ

Daniel Ferreira Schulz

Monografia apresentada como requisito parcial
para conclusão do Curso de Computação — Licenciatura

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

Prof.a Dr.a Edna Dias Canedo
CIC/UnB

Prof. Me. Edward Ribeiro
Senado Federal

Prof. Dr. Wilson Veneziano
Coordenador do Curso de Computação — Licenciatura

Brasília, 13 de junho de 2018

Dedicatória

Á minha família que sempre esteve presente me prestando apoio e incentivando o meu desenvolvimento.

Agradecimentos

Agradeço a todos os envolvidos que fizeram parte deste processo de formação. Agradeço aos meus familiares que me incentivaram a entrar na universidade e a me dedicar ao curso. Agradeço aos professores pela transmissão do conhecimento e por despertar o interesse nele. Agradeço aos meus amigos que tornaram a faculdade um ambiente mais interessante. Agradeço a equipe do BioNimbuZ e a minha orientadora pela disposição nas soluções dos problemas.

Resumo

O uso de nuvens computacionais tem crescido devido ao seu baixo custo e a pronta entrega dos recursos. Esse crescimento tem forçado uma disputa pelo mercado entre diversas empresas como Google, Microsoft, IBM, entre outras. Uma alternativa para se beneficiar das melhores ofertas é a federação de nuvens, onde os usuários podem escolher o provedor de recursos que melhor lhe atende. Assim, é firmado um Acordo de Nível de Serviço , de forma que o provedor deve garantir a Qualidade do Serviço (QoS). Para manter a Qualidade do Serviço, a federação deve prover um sistema tolerante a falhas, que implemente técnicas adequadas para específicos tipos de uso. Uma técnica que tem se mostrado muito eficaz em um ambiente de nuvens, é a técnica de *Checkpoint*. Dessa forma, para prover uma melhor experiência a usuários de nuvens, este trabalho propõe a integração de um serviço gerenciador de *checkpoint* a plataforma de federação de nuvens BioNimbuZ.

Palavras-chave: Computação em Nuvem, Tolerância a Falhas, Checkpoint-Restart

Abstract

The use of cloud computing has grown due to its low cost and the prompt delivery of resources. This growth has forced a market dispute between several companies like Google, Microsoft, IBM, among others. An alternative to taking advantage of the best deals is cloud federation, where users can choose the provider of resources that best suits them. Thus, a Service Level Agreement is established, so that the provider must guarantee Quality of Service (QoS). In order to maintain the Quality of Service, the federation must provide a fault-tolerant system that implements techniques that are appropriate for specific types of use. One technique that has proven very effective in a cloud environment is the Checkpoint technique. Thus, in order to provide a better experience for cloud users, this work proposes the integration of a checkpoint manager service with the BioNimbuZ cloud federation platform.

Keywords: Cloud Computing, Fault Tolerance, Checkpoint-Restart

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Com o avanço e progresso da humanidade, graças ao potencial tecnológico atingido, atualmente, quase a totalidade dos habitantes deste planeta fazem uso de serviços básicos como água, energia elétrica, gás, entre outros. Esses serviços estão disponíveis o tempo todo, prontos para suprir as necessidades dos clientes, utilizam, geralmente, o modelo pague pelo uso (*pay-per-use*), no qual os usuários pagam somente o que consumiram.

Nesta linha de raciocínio, o mesmo fenômeno se mostra presente com recursos computacionais. O conceito de nuvem aparece e se populariza graças a ampla utilização da internet. Acessar dados e executar aplicações na nuvem já é uma realidade, que tem crescido não somente para o uso pessoal, mas também para o uso comercial. Neste modelo, a complexidade da infraestrutura de TI é abstraída de forma que mais pessoas possam usufruir de seus frutos, sem a necessidade de conhecimento para configuração.

No cenário atual de utilização de serviços computacionais, aplicações científicas, tais como as de Bioinformática, tem se beneficiado do conceito de nuvem pela sua característica de tratar tarefas que exijam um alto poder de processamento, e inerentemente demandam significativas quantidades de recursos computacionais. Estas tarefas podem ser chamadas de *workflow*, que é um conjunto de diversas fases em que análises computacionais são executadas a partir de dados obtidos por meio de sequenciadores automáticos [11].

Desta forma, uma plataforma de nuvens federadas para a execução de diferentes aplicações, por intermédio de um ambiente de flexibilidade, eficiência, transparência e tolerancia à falhas, foi elaborada por Saldanha et al. [11]. Essa plataforma, chamada BioNimbuZ, permite o acesso a grande poder de armazenamento e processamento. A execução de *workflows* pode levar horas, em alguns casos, tempo suficiente para que falhas possam ocorrer. O surgimento de falhas de hardware ou software é inevitável, porém, seu tratamento para que o usuário não perceba, deve ser realizado. Utilizando técnicas de tolerância a falhas, alguns erros podem ser contornados sem que o usuário seja impactado. A técnica de *checkpoint* é uma alternativa para tolerância a falhas em ambientes de grande poder de

processamento.

Assim, este trabalho propõe a implementação da técnica de *checkpoint* na plataforma de nuvens federadas, BioNimbuZ.

1.1 Motivação

A execução de aplicações em um ambiente de nuvens se apresenta como um conceito capaz de integrar diferentes infraestruturas, apto a proporcionar uma visão na qual a quantidade de armazenamento e de processamento seja vista como recursos ilimitados. Todavia, a utilização dos recursos deve ser realizada de maneira eficaz para, assim, garantir a tolerância a falhas nas tarefas executadas, principalmente nas que possuem longa duração.

Diante da necessidade de tornar tarefas executadas na nuvens mais tolerantes a falhas, a motivação deste trabalho é implementar a técnica de *checkpoint* na plataforma de federação BioNimBuZ, de maneira que seu funcionamento seja automático e, deste modo, mantenha salvo os estados válidos já processados.

1.2 Problema

Tarefas de Bioinformática podem exigir um longo tempo de processamento, o que gera um gasto financeiro significativo, mesmo em um ambiente de nuvens. Atualmente na plataforma de federação BioNimbuZ, uma falha em uma das tarefas de um *workflow* pode causar a perda de todo o processamento já realizado, além da não finalização da tarefa corretamente.

Nesta conjuntura, este trabalho propões a implementação da técnica de *checkpoint* como uma alternativa de tolerância a falhas.

1.3 Objetivo

1.3.1 Objetivo Geral

O objetivo principal deste trabalho é implementar a técnica de *checkpoint*, de forma que seu funcionamento seja automático, onde os estados válidos das tarefas executadas sejam salvos, e sejam reexecutados agilmente em caso de uma interrupção, consequentemente, garantir a resiliência do sistema e prover meios para reduzir o tempo de execução das tarefas em caso de falhas.

1.3.2 Objetivos Específicos

Para cumprir o objetivo geral desde trabalho, faz-se necessário atingir os seguintes objetivos específicos:

- Implementar a técnica de reenvio de tarefas na plataforma BioNimbuZ;
- Implementar a técnica de *checkpoint*, por meio da aplicação DTMCP;
- Mensurar o impacto do uso do *checkpoint* em aplicações de Bioinformática;
- Mensurar os benefícios do uso do *checkpoint* em aplicações de Bioinformática.

1.4 Metodologia Científica

A metodologia de pesquisa utilizada foi a exploratória quantitativa, visando investigar o tema abordado. A metodologia proposta foi dividia em três etapas, de maneira a facilitar o entendimento do trabalho.

- **Etapa 1:** Na primeira etapa foi realizado o referencial bibliográfico, visando aprofundamento dos conceitos e melhor entendimento do assunto.

- **Etapa 2:** Na segunda etapa foram realizados testes do uso da técnica de *checkpoint* em diferentes cenários, com o objetivo de coletar dados para avaliar o seu desempenho.

- **Etapa 3:** Na terceira e última etapa, foram avaliados os dados dos testes e calculado o desempenho do uso do *checkpoint*, de maneira a validar as hipóteses iniciais.

1.5 Estrutura da Monografia

Este trabalho contém, além deste capítulo introdutório, mais cinco capítulos. No Capítulo 2 são apresentados os conceitos de sistemas distribuídos, seu histórico e a definição de conceitos como *clusters, grids* e nuvens. Também é abordado a evolução dos sistemas distribuídos para o conceito de federação de nuvens.

O Capítulo 3 aborda os conceitos de tolerância a falhas, como erro, falha e defeito. São detalhadas as principais técnicas de tolerância a falhas, inclusive o *checkpoint*.

O Capítulo 4 aborda o conceito de *workflow* científico, suas características e ciclo de vida. Também é detalhada a plataforma de federação de nuvens BioNimuZ, pois, apresenta-se sua arquitetura, principais características e funcionalidades.

O Capítulo 5 exibe o trabalho realizado, como uma nova alternativa de tolerância a falhas. Neste capítulo é detalhada a implementação do trabalho, *workflow* utilizado e os testes efetuados.

Por fim, no Capítulo 6 são apresentadas as considerações finais obtidas com a realização desta pesquisa, assim como alguns trabalhos futuros

Capítulo 2

Referencial Teórico

O objetivo deste capítulo é apresentar a revisão bibliográfica dos temas relacionados ao trabalho redigido nesta monografia. Assim, inicialmente é introduzido na Seção 2.1 o surgimento de sistemas distribuídos. Na Seção 2.2 é abordado o conceito de nuvem computacional, suas características, sua arquitetura e taxonomia. Por fim, na Seção 2.3 são mostrados os detalhes de federação de nuvens.

2.1 Histórico

O uso dos computadores e, nos últimos anos, os celulares tem crescido rapidamente, para atender as necessidades de milhares requisições. Assim, um único servidor não é suficiente, mesmo que esse servidor possua vários processadores de última geração, uma grande quantidade de memória, e que suas aplicações utilizem as melhores técnicas de programação. Para suprir essa necessidade, surgiu o que é chamado de sistemas distribuído, que conforme Tanenbaum [12] é "Um conjunto de computadores independentes que se apresenta a seus usuários como um sistema único e coerente", ou seja, distribuir o trabalho para diversos computadores de forma que estes sejam organizados a se comportarem como um só, tanto para o usuário quanto para os programas que neles são executados. Todo esse processo é possível através do *middleware*, um software que se situa no topo do sistema operacional, responsável pelo relacionamento em grupo dos computadores de modo consistente. Além do relacionamento em grupo, o *middleware* permite que diferentes hardwares e sistemas operacionais se comuniquem, aumentando a heterogeneidade dos componentes. O primeiro modelo de sistemas distribuídos criado foi o *cluster*, apresentado na Seção 2.1.1.

2.1.1 Cluster

Na década de 60 a IBM [13] foi pioneira na criação de *clusters* na tentativa de unir *mainframes* de forma a aumentar o paralelismo de suas tarefas, porém, devido as limitações da época na área de redes, protocolos de computação distribuída, e microprocessadores, o *cluster* não se mostrou vantajoso [14]. Apenas em 1994 com os avanços da tecnologia o *cluster* foi bem visto pela sociedade acadêmica, assim, conseguindo resultados que justificassem seu desenvolvimento pelo projeto Beowulf. Um *cluster* é caracterizado por dois ou mais computadores, como *mainframes*, servidores e estações de trabalho, situados próximos geograficamente uns dos outros, ou seja, no mesmo *datacenter*, onde são ligados por uma rede de alta performance, assim como apresentado na Figura 2.1. Outra características dos *clusters* é a sua homogeneidade de hardware e de software entre seus componentes [12].

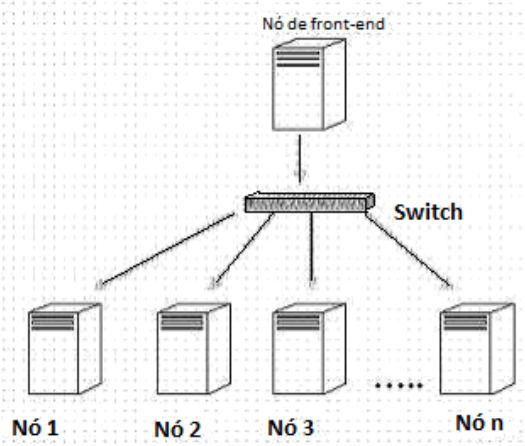

Figura 2.1: Exemplo de um Cluster.

2.1.2 Grid

Na década de 90 cientistas desenvolveram outro modelo de sistemas distribuído, o qual mascarava sua infraestrutura para o usuário, assemelhando ao que acontece com a energia elétrica, no qual uma pessoa utiliza apenas uma tomada e o serviço (energia) é fornecido sem que ela saiba como. Esse modelo é chamado de *grid* computacional [15]. Assim, por meio de computadores virtuais os componentes de hardware e software do *grid* podem ser diferentes um dos outros, o que caracteriza sua heterogeneidade. Com os avanços na área de redes, a comunicação entre os computadores é de alta velocidade, mesmo estes se situando em diferentes pontos do globo.

2.1.3 Nuvem Computacional

Atualmente, a informatização é uma necessidade de muitas empresas, mesmo um supermercado que em toda sua história possuiu apenas lojas físicas, agora, precisa oferecer seu catálogo de produtos, promoções e até realizar vendas através de um *site* na Internet, além de conhecer seus clientes armazenando suas informações e transações em um banco de dados. Além disso, por meio do processamento destes dados recomendar novas possibilidades de compras via e-mail. Todo esse processo exige uma gama de recursos computacionais, como um servidor de hospedagem, sistemas operacionais, programas de aplicação, acesso a rede entre outros. Em alguns casos os gastos e conhecimento destes recursos somados a serviços de manutenção, configuração e monitoramento podem ser um problema, surgindo assim uma oportunidade no mercado para terceirização de serviços de infraestrutura de computação.

Com o aprimoramento da computação em *grid* e o amplo uso da Internet surge a nuvem (*cloud computing*), um modelo de computação que oferece o aluguel de recursos computacionais em forma de serviço através da Internet. Nesse modelo uma empresa que possui em seu *dacenters* uma grande quantidade de recursos computacionais, como servidores e discos, os interliga através de uma rede de alta velocidade e os virtualiza, de forma que agora podem ser fornecidas parcelas ou conjuntos desses recursos ao usuários. Por outro lado, o usuário por meio da Internet solicita os recursos desejados, que são disponibilizados imediatamente e cobrados de acordo com o uso. Assim, para criar um *site* com um banco de dados não há a necessidade de comprar um servidor, adquirir licenças de software, um pacote de redes, preparar o ambiente físico e configurá-los, todos estes podem ser fornecidos pela nuvem de maneira instantânea [12]. Outro ponto importante é que a soma de todos esses recursos pode sair bem caro caso o usuário opte por comprá-los, principalmente, para quem está iniciando, como *startups*, porém, na nuvem, não há investimento inicial, sendo cobrado de acordo com o número de acessos, a quantidade de espaço e o processamento utilizados, além do tempo que os serviços estão disponíveis.

Há diversas definições das principais características de uma nuvem, segundo o Instituto Nacional de Padrões e Tecnologia [16], essas são as cinco características essenciais:

• **Serviço automático de acordo com a demanda** os usuários podem se abastecer dos recursos fornecidos instantaneamente sem a necessidade de interação humana;

• **Acesso amplo pela rede** os recursos são fornecidos pela Internet de forma padronizada, permitindo a heterogeneidade dos dispositivos dos usuários;

• ***Pooling* de recursos** os recursos estão disponíveis para diversos usuários com capacidade de alocação e realocação dinamicamente, e a localização destes recursos é abstraída para o usuário;

• **Elasticidade rápida** os recursos podem ser alocados e liberados de maneira elástica, imediatamente a requisição do usuário;

• **Serviço mensurado** o uso dos recursos é calculado de forma transparente e repassado ao consumidor, conforme previamente acordado com o provedor.

2.1.4 Arquitetura

Segundo Foster [1], a arquitetura de uma nuvem pode ser divida em quatro camadas, as quais são hardware, infraestrutura, plataforma e aplicação, apresentadas na Figura 2.2, e detalhadas a seguir:

• **Hardware:** Essa camada é composta pelos recursos computacionais, como os recursos de processamento, de armazenamento e de rede. Ela é implementada em *datacenters* onde milhares de servidores são organizados em *racks*, estes acessando as controladoras de disco por meio de uma rede composta por *switches* e roteadores. O seu gerenciamento envole a configuração dos dispositivos, manutenção da rede e do ambiente físico;

• **Infraestrutura:** Nesta camada os componentes de hardware da camada inferior são abstraídos/encapsulados por meio de virtualização, no qual os usuários finais e as camadas superiores os enxergam como componentes integrados, como um sistema de arquivos lógico, um computador virtual, um banco de dados e, assim por diante. Essa camada é essencial para oferecer características chaves que só podem ser disponibilizadas por meio da virtualização, como a atribuição dinâmica de recursos;

• **Plataforma:** Nesta camada é adicionado um conjunto de programas para a criação de uma plataforma de desenvolvimento ou execução. Dentre estes programas podem ser citados os sistemas operacionais, os *middlewares*, o serviço de agendamento. O principal objetivo desta camada é diminuir o tempo de instalação desses programas que podem ser pré instalados de acordo com a demanda do usuário;

• **Aplicação:** No topo da arquitetura tem-se as aplicações que são executadas na nuvem. A sua vantagem é utilizar os benefícios das camadas anteriores como provisionamento de recursos de forma automática, afim de obter um maior desempenho e abstração de todas as configurações anteriores.

2.1.5 Modelos de Serviço

As nuvens podem ser classificadas de acordo com o seu modelo de serviço, ou seja, o tipo de serviço que a nuvem oferece. Tanto na literatura quanto no mercado é possível encontrar diversos tipos de modelos, porém, os três mais utilizados são [17] Infraestrutura como Serviço, Plataforma como Serviço e Software como Serviço, definidos a seguir [12] e apresentados na Figura 2.3:

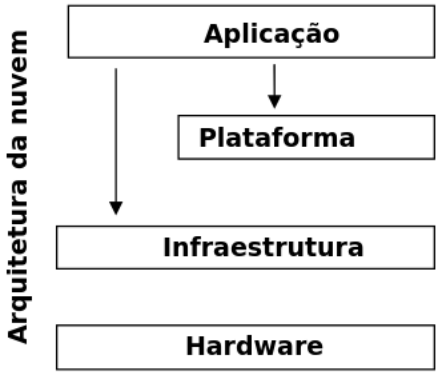

Figura 2.2: Arquitetura de uma Nuvem, adaptada de [1].

• ***Infrastructure-as-a-Service*** **(IaaS):** Fornece recursos de infraestrutura aos usuários tais como poder computacional, armazenamento de dados, redes virtuais e até máquinas completas. Neste modelo a virtualização é essencial para que os usuários compartilhem máquinas físicas, pagando apenas pelo o que é utilizado e quando necessário aumentar ou diminuir os recursos de acordo com a necessidade do usuário. Alguns exemplos de serviços deste modelo são: Amazon EC2 [18] e o Google Compute Engine [19];

• ***Plataform-as-a-Service*** **(PaaS):** Fornece um ambiente pronto para que o usuário possa programar, criar aplicações e serviços como um sistema operacional, banco de dados e um servidor web pré-instalados; sem a necessidade de softwares terceiros. Além dos programas oferecidos, os recursos de infraestrutura ficam abstraídos ao usuário que não precisa configurá-los. Exemplos de provedores deste serviço são: Microsoft Azure [20], IBM Bluemix [21] e Google App Engine [22];

• ***Software-as-a-Service*** **(SaaS):** Fornece aplicações aos usuários, estas sendo executadas pelo provedor onde são acessadas através da Internet. Este modelo possui um baixo custo para o usuário, pois ele só paga por aquela aplicação que usa e não possui nenhum trabalho para manutenção deste serviço, que é de responsabilidade do provedor. Exemplos deste tipo de modelo de serviço são: Google Docs [23] e Microsoft Office 365 [24];

2.1.6 Modelos de Implantação

Um dos principais impasses para a utilização das nuvens é a questão da segurança dos dados, pois eles podem ser armazenados em um ambiente físico fora do controle do usuário e sob administração de outra empresa. Em alguns casos como bancos, a segurança é o principal pilar da instituição, sendo necessária formas diferentes de implementação da nuvens. Para restringir o acesso a nuvem aos interesses dos usuários, existem quatro formas diferentes de implantação que serão apresentadas a seguir:

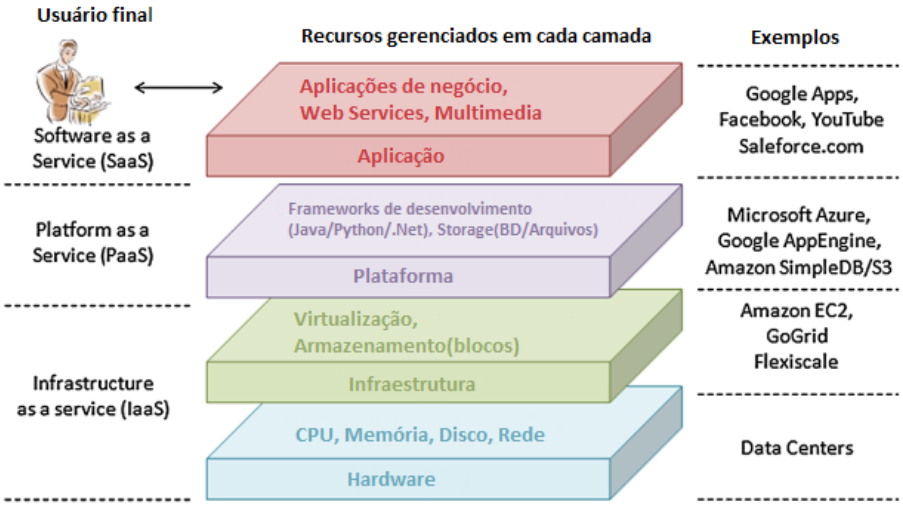

Figura 2.3: Modelos de Serviço de uma Nuvem, adaptados de [2].

• **Pública:** Seus serviços são disponibilizados para o público em geral, não havendo restrições. A nuvem é administrada por uma grande empresa que reparte os recursos de seu *datacenter* de forma virtualizada para os usuários no modelo de comercialização, no qual ele paga o quanto utilizar, e acessa os serviços por meio da Internet. Não possui exigência em relação a localidade dos recursos;

• **Privada:** É a nuvem que está inserida em um contexto empresarial no qual apenas funcionários possuem acesso. Seu gerenciamento é feito pela própria instituição ou por uma empresa administradora. O motivo deste modelo é o aumento da segurança do dados onde estes ficam restritos a um pequeno público, e dentro do *firewall* da instituição. Uma de suas vantagens, além da segurança, é a largura de banda da rede visto que a nuvem é implantada dentro da organização;

• **Comunitária:** Nuvem que atende aos interesses de um grupo de organizações com um fim em comum (área jurídica, econômica ou mesmos níveis de qualidade de serviço), sendo que seus funcionários, são os usuários dos serviços da nuvem. Pode ser implantada e gerenciada por uma ou mais dessas empresas;

• **Híbrida:** Ambiente no qual há a junção de dois dos modelos ou mais dos modelos anteriores. Utilizado para abranger mais recursos a uns dos modelos, como unir um banco de dados de uma nuvem privada com a aplicação de uma nuvem pública.

2.2 Nuvens Federadas

Mais de dez anos se passaram desde o surgimento do conceito da nuvem computacional. Assim, como sua evolução, as necessidades também cresceram, sua utilização se tornou

mais popular e novas estratégias surgiram, como a federação de nuvens. A federação de nuvens pode ser definida como a forma de integrar mais de uma nuvem de um mesmo ou de provedores diferentes para um melhor compartilhamento dos recursos [25]. Visto que uma única nuvem ao ser impactada por alguma indisponibilidade, milhares de clientes podem ser afetados. Dessa maneira, a fim de aumentar a tolerância a falhas, algumas empresas espalharam seus *datacenters* pelo globo, utilizando técnicas de redundância, com o objetivo de melhorar o acordo de nível de serviço. Assim, unir nuvens de forma a oferecer um serviço integrado acabou se tornando uma necessidade. Seguindo a mesma ideia da computação em *grid* os serviços de nuvens federadas devem ser comparados ao fornecimento de água e energia elétrica, no qual todo o processo é abstraído e o consumidor só é cobrado pelo que usa.

Outro ponto importante na federação de nuvens é o melhor aproveitamento de recursos, visto que em poucos períodos, picos de uso podem sobrecarregar um determinado serviço, logo, para que a Qualidade do Serviço (Quality of Service - QoS) não seja prejudicada, um provedor pode utilizar recursos de outras nuvens, garantindo o nível de serviço acordado [26]. O mesmo acontece quando uma nuvem possui baixa utilização, podendo oferecer seus recursos ociosos para outras nuvens, aumentando seu lucro. A federação de nuvens também pode ser utilizada para unir nuvens privadas de uma empresa a uma nuvem pública, caracterizando nuvens híbridas, conforme descrito na Seção 2.2.3.

Contudo, realizar a integração de nuvens pode gerar desafios, devido a comunicação entre diferentes programas de gerenciamento, hardwares, a fim de prover um serviço de forma consistente. Então, para que haja a federação alguns requisitos são necessários, conforme [27]:

• **Automatismo e escalabilidade:** Por meio de mecanismos de descoberta o software gerenciador de uma nuvem federada deve ser capaz de identificar entre as nuvens disponíveis a que melhor atende à sua necessidade, sendo capaz de se adaptar as mudanças;

• **Segurança interoperável:** As nuvens devem possuir integração entre as diversas tecnologias de segurança, de maneira que uma nuvem não precise alterar sua política para se unir a federação.

2.2.1 Tipos de Arquitetura

Nuvens federadas podem ser classificadas em duas maneiras, de acordo com a organização de sua arquitetura. Segundo Buya [3], os tipos são apresentados na Figura 2.4 e detalhados a seguir:

• **Centralizada:** Baseada em um unidade central chamada de Troca de Nuvem - TN *(Cloud Exchange)*, no qual essa unidade centralizadora atua como um mercado, fornecendo

ofertas e demandas de recursos. Cada nuvem possui um programa agente que recolhe estatísticas de saúde e informações a respeito da disponibilidade dos recursos, os envia a central, que atualiza os registros periodicamente. O provedor de cada nuvem também fornece a TN informações como a localidade dos centros de dados e os valores de seus recursos. Com os dados recolhidos, a TN pode executar algoritmos que identificam os recursos com o melhor preço que atendam a um determinado Acordo de Nível de Serviço (termo em inglês - SLA);

• **Ponto-a-ponto:** Baseada na comunicação direta entre as nuvens, sem a necessidade de uma unidade centralizadora. Nesta arquitetura, nuvens realizam a negociação dos recursos disponíveis, àqueles que desejam adquirir, por meio de uma camada de interface. Cada nuvem realiza o gerenciamento das informações oferecidas pelas nuvens parceiras, cabendo a própria nuvem aplicar regras para aluguel e o empréstimo dos recursos.

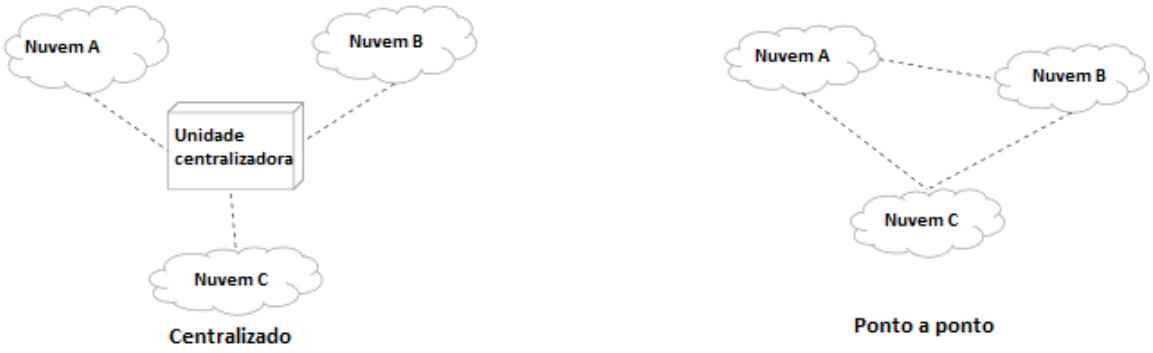

Figura 2.4: Arquiteturas de Federação de Nuvens, adaptado de [3].

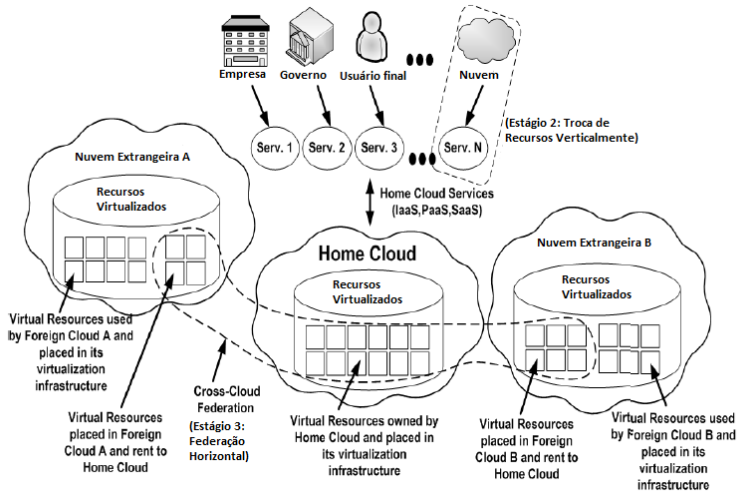

Figura 2.5: Cenário de Integração de Nuvens, adaptado de [4].

Conforme visto ao longo deste capítulo, as nuvens federadas são o próximo passo na evolução dos sistemas distribuídos que ainda passa por uma constante evolução. Segundo Bittman [4], a evolução da federação de nuvens pode ser definida em três fases. A primeira caracteriza ilhas de centros de dados proprietários que não se comunicam com os demais. A segunda fase é caracterizada pelo crescimento vertical da nuvens, onde a empresa integra os seus *datacenters*, que é a fase atual. A última e terceira fase apresenta o crescimento horizontal das nuvens, na qual estas se relacionam com nuvens de diferentes empresas, conforme apresentado na Figura 2.5.

2.3 Considerações Finais

Neste capítulo foram detalhadas as informações a respeito da evolução da computação distribuída, desde o início dos *clusters* até a federação de nuvens. Muitos desafios envolvem esse processo evolutivo, principalmente, entregar um serviço tolerante a falhas, que será o assunto do próximo capítulo.

Capítulo 3

Tolerância a Falhas

Objetivo deste capítulo é apresentar os conceitos de tolerância a falhas em um ambiente de nuvens federadas. Assim, na Seção 3.1 é abordado como se origina uma falha, sua evolução para um erro e, consequentemente, o defeito gerado. Na Seção 3.2 são apresentadas técnicas de tolerância a falhas e, por fim, na Seção 3.3 é detalhado o *checkpoint-restart*, que é a técnica utilizada neste estudo.

3.1 Definições

Uma falha pode ser definida como a incapacidade de um sistema de realizar o que lhe foi requisitado, devido a um estado anormal ou um *bug*, em uma ou mais partes de um sistema. Assim, a falha está mais relacionada ao hardware [5]. Segundo Bilal [28], uma falha pode ser classificada em transiente, intermitente e permanente, e as duas falhas mais comuns em ambiente de nuvem são as falhas arbitrárias e as falhas de falência.

Por outro lado, um erro é a consequência de uma falha em nível de sistema. Em tolerância a falhas é conhecido que as falhas ocorrem, assim, é realizada a tentativa de identificar os erros e tomar providencias para prevenir e minimizar os defeitos [29].

O defeito é um estado no qual um serviço não desempenha o objetivo desejado ou pretendido. Ele refere-se a má conduta de um sistema que pode ser observada por um usuário, que pode ser um ser humano ou outro sistema de computador. Várias eventos podem dar errado em um sistema, mas o defeito só ocorre se sua saída for um resultado incorreto [5]. Na Figura 3.1 é demonstrado o fluxo de uma falha até um defeito, e na Figura 3.2 é demonstrado os níveis que se encontram as falhas, os erros e os defeitos. Os defeitos mais comuns segundo Tanembaum [30] são:

Defeitos Arbitrários: Também podem ser chamados de defeitos bizantinos, nesse tipo de defeito, o sistema apresenta inconsistência em mais de um dos seus componentes, causando um comportamento inesperado. O sistema pode receber e processar as requisi-

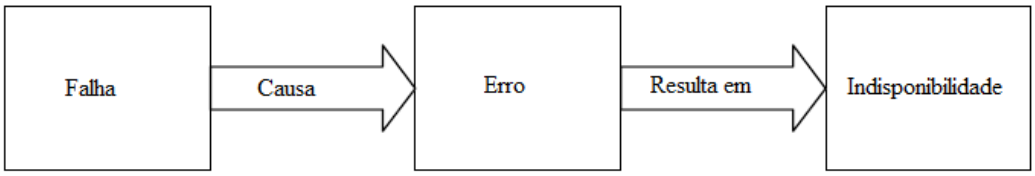

Figura 3.1: Origem de um Defeito, adaptado de [5].

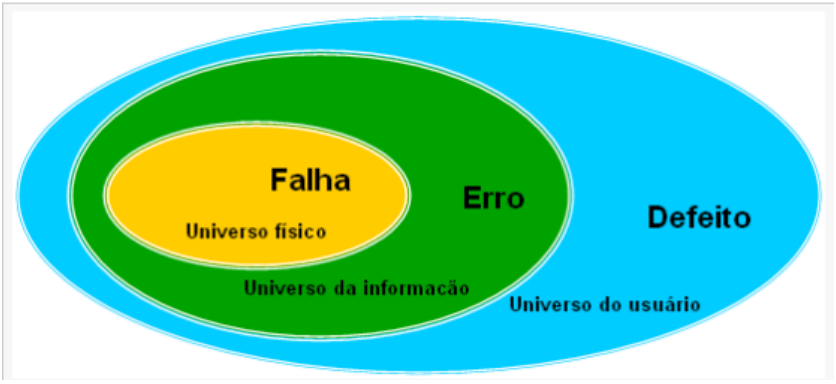

Figura 3.2: Universo da Falha, do Erro e do Defeito.

ções, porém suas saídas podem apresentar divergências. É difícil de ser detectado pois o sistema continua respondendo, mesmo que incorretamente.

• **Defeitos de Travamento:** Neste tipo de defeito os componentes do sistema apresentam parada total, causando no não processamento das requisições e desligamento do sistema;

Defeitos de Omissão: Ocorrem quando um sistema não é capaz de responder a uma requisição. Pode ser causado por um problema no meio de transporte, impossibilitando a sua chegada ao servidor, ou no envio ao usuário;

Defeitos por Temporização: Ocorrem quando uma resposta ultrapassa o tempo limite da requisição. Frequentemente relacionado a problemas de desempenho de processamento ou de redes. Um erro bastante conhecido é o *time-out*, utilizado para que uma requisição não fique aberta por tempo indeterminado.

• **Defeitos de Resposta:** Ocorrem quando a resposta de uma determinada requisição está incorreta. Essa falha pode estar relacionada ao valor da resposta, que possui um conteúdo divergente ao esperado ou ao seu estado. Um exemplo de falha de estado de resposta ocorre quando a requisição não é reconhecida.

3.2 Técnicas de Tolerância a Falhas

Tolerância a falhas é a propriedade de um sistema, que permite que um serviço seja disponibilizado, mesmo que com uma queda de performance, quando alguns de seus componentes apresentam uma ou mais falhas [5].

Amin [6] classifica as técnicas de tolerância a falhas em proativas e reativas, conforme apresentadas na Figura 3.3, e descritas nas próximas seções.

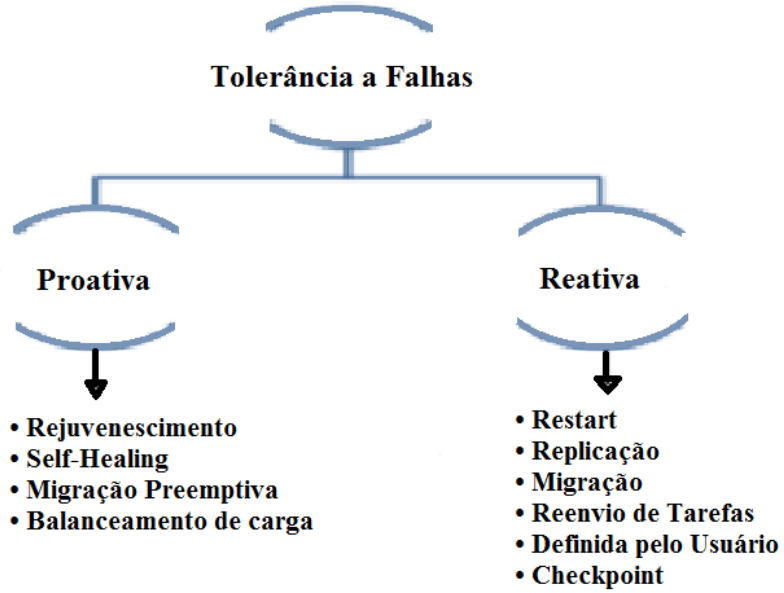

Figura 3.3: Tipos de Tolerância a Falhas, adaptado de [6].

3.2.1 Proativas

Técnicas proativas de tolerância a falhas se baseiam em uma monitoração proativa para prever possíveis falhas antes que elas ocorram, assim, substituindo o componente suspeito por um novo sem falhas, antes de qualquer problema [5]. Assim, as principais técnicas deste tipo são:

• **Rejuvenescimento:** Nesta técnica um sistema é agendado para ser periodicamente reiniciado, assim, seu software se inicia de um estado novo, livre de falhas [6]. Um exemplo são programas que utilizam bastante memória e podem apresentar lentidão a longo prazo.;

• ***Self-Healing*:** Uma tarefa pode ser dividida em mais de uma parte para aumentar sua performance. Assim, quando diversas instâncias de uma aplicação estão rodando em diferentes máquinas virtuais, a aplicação resolve falhas de suas instancias automaticamente [31];

• **Migração Preemptiva:** Nessa técnica a aplicação é frequentemente observada e analisada. Caso haja algum indício de uma possível falha, o estado da tarefa é salvo e sua execução continua em outro recurso [6];

• **Balanceamento de Carga:** Assim que um determinado recurso atinja um limiar máximo, a tarefa pode ser distribuída para outros recursos, impedindo o esgotamento antes que ele ocorra [5]. Também pode ser utilizado para dividir as tarefas igualmente entre os recursos para que um deles não se sobrecarregue.

3.2.2 Reativas

Técnicas reativas são utilizadas quando um defeito já ocorreu no sistema [5]. Sua utilização é para ajudar o sistema em um estado defeituoso a se recuperar para um estado estável, para que assim suas tarefas voltem a serem executadas produzindo os resultados esperados. Dessa forma, o tempo de indisponibilidade é reduzido [29]. Algumas dessas técnicas são descritas a seguir [27]:

• ***Restart*:** Esta técnica é implementada em nível de aplicação. Caso uma tarefa não finalize em um determinado tempo previsto, essa tarefa é abortada e reiniciada. A tarefa de definição do tempo limite para a execução é muito importante, pois se for calculado errado, a tarefa pode ser abortada um pouco antes de finalizar [5];

• **Replicação:** É uma das técnicas de tolerância a falhas mais populares. As tarefas podem ser executadas em mais de um recurso, assim, caso um dos recursos apresente algum erro, as demais tarefas podem ser finalizadas com sucesso [29]. Também pode ser utilizado para armazenamento de dados em diferente locais, no caso do acesso à algum deles falhar, outro pode ser utilizado. A replicação utiliza redundância, assim, pode ser implementada utilizando ferramentas como o *Hadoop* e o *AmazonEc2* [6];

• **Migração:** Em caso de falha em alguma tarefa, esta pode ser enviada para re-execução em outra máquina. Essa técnica pode ser implementada utilizando *HAProxy* [32];

• **Reenvio de Tarefas:** Nessa técnica, após verificado que houve uma falha, a tarefa pode ser enviada para reexecução no mesmo recurso que houve a falha, ou em um novo recurso, de maneira que o fluxo de execução não seja interrompido [29];

• **Definida pelo Usuário:** Para um caso específico de falha, o usuário pode definir como agir para que continue a execução das tarefas [29];

• ***Checkpoint*:** Devido a importância da técnica de *checkpoint* nesse estudo, ela será detalhada separadamente, na Seção 3.3.

Conforme o crescimento das nuvens, em tamanho e em complexidade, é de grande criticidade garantir a estabilidade, a disponibilidade, e a confiabilidade nesses sistemas. Para garantir esses requisitos, e mensurar as falhas e indisponibilidades, provedores de

nuvens têm adotado diversos mecanismos que implementam tolerância a falhas em nível de sistema [28].

3.3 *Checkpoint-Restart*

Algumas tarefas com longa duração, podem ser um problema para o *SLA*, mesmo que sejam implementadas as técnicas apresentadas anteriormente, pois por exemplo, no caso de uma tarefa que demore vinte e quatro horas para finalizar, e após vinte e três apresente um erro, todo o esforço de quase um dia pode ser perdido. Para solucionar esse problema, a técnica de *checkpoint* foi proposta e implementada na nuvem.

O mecanismo do *checkpoint* salva o estado de uma tarefa periodicamente de acordo com um intervalo de tempo previamente definido. O estado da tarefa é armazenado em um dispositivo não volátil, como disco rígido, e fica disponível caso alguma indisponibilidade ocorra. Caso uma tarefa falhe, ela é reexecutada a partir do último *checkpoint* efetuado, de maneira que a maior parte da execução é reaproveitada, na maioria dos caso, sendo essa, a maior vantagem da técnica de *checkpoint-restart* [8].

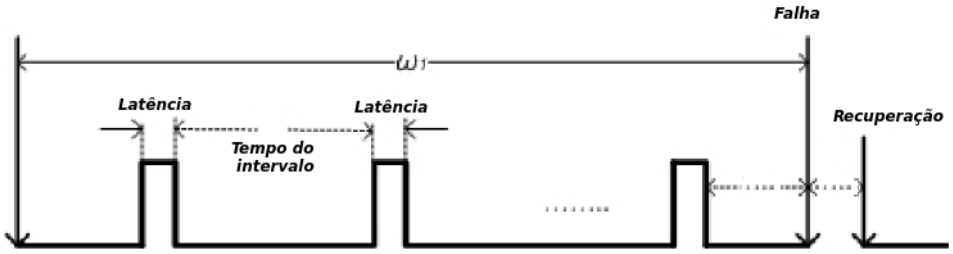

Figura 3.4: Comportamento de um Modelo de *Checkpoint*, adaptado de [7].

O *checkpoint* realiza basicamente um *snapshot* do processo, armazenando todas as informações necessárias para que o processo possa ser reexecutado a partir do ponto que parou. No caso de um processo que utiliza muita informação, como diversas variáveis, ocupando grande espaço em memória, o tamanho a ser gravado será correspondente ao utilizado pelo processo. Assim, utilizar *checkpoints* resulta no aumento do tempo de execução de uma tarefa, esse aumento é chamado de *checkpoint overhead*. A latência é o tempo para realizar a gravação do *checkpoint* em disco, normalmente, suas somas resultam no mesmo tempo do *overhead*. Logo, quanto menor o número do intervalo de tempo entre os *checkpoints*, maior será o tempo do *overhead* [33].

Uma linha de pesquisa que está em constante aprimoramento é a avaliação do tempo de intervalo ideal. Uma estratégia para reduzir o tempo de latência é realizar o *checkpoint* incremental, que aproveita as páginas não alteradas em vez de salvar todo o processo

todas as vezes [5]. Nesse sentido, é demonstrada na Figura 3.4 o comportamento de um modelo de *checkpoint*.

3.3.1 Implementação de *Checkpoint*

Existem diferentes formas de implementar a técnica de *checkpoint*, que se referem a forma de integrar a aplicação com a plataforma. Nesta seção serão abordadas as três principais técnicas segundo Egwutuoha [8], que são apresentadas na Figura 3.5:

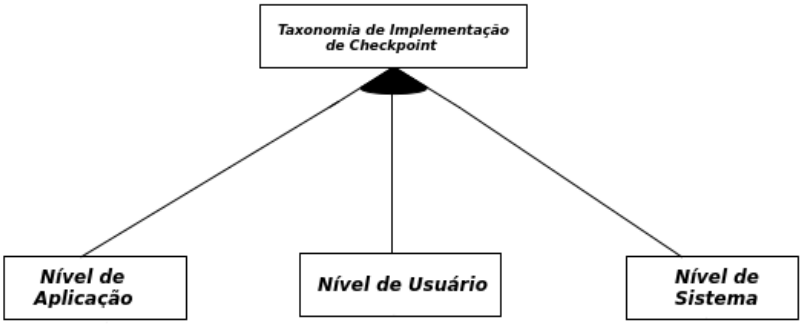

Figura 3.5: Taxonomia de Implementação de *Checkpoint*, adaptado de [8].

Nível de aplicação: Neste nível de implementação o código do *checkpoint* é inserido junto ao código da aplicação. As atividades de *checkpoint* são gerenciadas pela aplicação. Basicamente, envolve a inserção de código onde uma quantidade pequena de estados precisa ser salva. As suas desvantagens são o uso limitado em sistemas heterogêneos e a falta de transparência para o usuário.

Nível de usuário: Utiliza uma biblioteca de nível de usuário e normalmente não é transparente para o usuário. A sua principal desvantagem é a limitação das chamadas de sistemas que podem ser implementadas, como exemplo algumas aplicações não podem realizar *checkpoint* devido a permissionamento do sistema.

Nível de sistema: É sempre transparente ao usuário e não necessita modificação da aplicação. Aplicações podem ser salvas a qualquer momento conforme um intervalo previamente definido. Dessa maneira, pode ser difícil essa implementação devido ao fato de nem todos os fornecedores de sistemas operacionais disponibilizarem seu código fonte. Logo implementações em nível de sistema não são portáveis.

3.4 Considerações Finais

Neste Capítulo foram detalhadas as informações a respeito de tolerância a falhas, os conceitos de falha, erro e defeito, até as técnicas utilizadas para minimização destes. Muitos

desafios envolvem esse processo, principalmente, integrá-los a plataformas de federação, tais como o BioNimbuZ, que será apresentado no próximo capítulo.

Capítulo 4

Plataforma de Federação BioNimbuZ

O objetivo deste capítulo é apresentar as principais características da plataforma de federação de nuvens BioNimbuZ, na qual será implementada a técnica de *checkpoint* no serviço de tolerância a falhas, desenvolvida neste estudo. Para isso, na Seção 4.1 são introduzidos os conceitos de *workflow* científico, que será utilizado para realização dos testes. A Seção 4.2 aborda as principais características do BioNimbuz, aspectos da plataforma, seus objetivos e uma visão geral de seu funcionamento. Na Seção 4.3 será apresentada a arquitetura do BionimbuZ, suas camadas e seus serviços. E por fim, a Seção 4.4 é reservada para o serviço de tolerância a falhas.

4.1 *Workflow* Científico

O conceito de *workflow* surgiu na década de 70, devido aos processos de automação de escritório, com o objetivo de oferecer soluções para diminuir a geração e a distribuição de documentos em papel em uma organização. Assim, um *workflow* pode ser interpretado como a automação total ou parcial de um processo, na qual informações ou tarefas são passadas de uma entidade para outra de acordo com um conjunto de regras [34].

Recentemente, o conceito de *workflow* vem sendo aplicado no meio científico na automação de experimentos que utilizem grande poder computacional e manipulam uma grande quantidade de dados, possivelmente, distribuídos. Nesse contexto, um *workflow* pode ser definido como um grafo direcionado acíclico, no qual seus vértices e arestas representam suas atividades e dependências, respectivamente [35].

No contexto da Bioinformática, as análises computacionais dos dados, obtidos por meio de sequenciamento automático são realizadas em diferentes fases ou passos. Para cada fase existe um conjunto de ferramentas de Bioinformática a serem utilizadas. Entretanto, cada tipo de pesquisa resulta numa combinação diferente de ferramentas, de acordo com as

necessidades da pesquisa, e este fluxo de passos é chamado de *workflow* de Bioinformática [9].

Do ponto de vista computacional, os biólogos inserem um arquivo de entrada que passa por diversas etapas de processamento, podendo estas serem a Filtragem, o Mapeamento ou a Montagem, a Análise, e outros. Em cada etapa, um novo arquivo é gerado como saída e pode ser inserido na próxima etapa, no fim, o arquivo de saída é gerado com os resultados finais de um *workflow*. Na Figura 4.1 é demonstrado um exemplo de um *workflow* de Bioinformática.

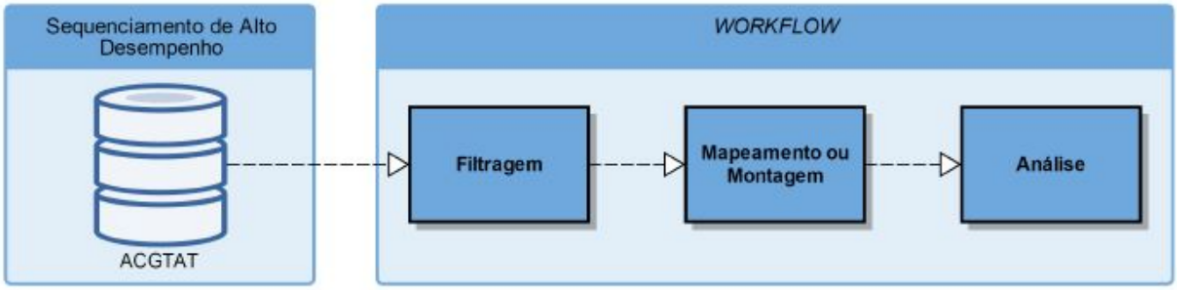

Figura 4.1: Exemplo de Workflow de Bioinformática [9].

Assim, gerenciar *workflows* de Bioinformática pode não ser uma tarefa simples, devido a algumas tarefas que exigem muitos recursos. Por isso, será apresentada na Seção 4.2 o BioNimbuZ, ferramenta utilizada neste trabalho que implementa de maneira transparente um ambiente de nuvem federada.

4.2 Principais Características

O BioNimbuZ é uma plataforma para federação de nuvens híbridas, que foi desenvolvida primeiramente por Saldanha [11], e tem sido constantemente aprimorada por outros trabalhos [9], [10], [36], [37], [38]. O objetivo do BioNimbuZ é suprir as necessidades de plataformas de nuvens federadas, visto que nuvens isoladas não conseguem atender, em muitos casos, necessidades de execução de aplicações de Bioinformática que podem demandar um alto poder de processamento e de armazenamento em geral.

A plataforma BioNimbuZ permite a federação de nuvens de diversos tipos, tanto públicas quanto privadas. Assim, cada provedor pode implementar suas próprias políticas internas, além de oferecer ao usuário transparência e a ilusão de recursos infinitos. Desta forma, o usuário pode usufruir de diversos serviços sem se preocupar com o provedor que está fornecendo os recursos. Por meio de *plugins* de integração, que se encarregam de mapear as requisições do BioNimbuZ para as requisições de cada provedor, é flexibilizado a inclusão de novos provedores.

Originalmente, o BioNimbuz foi implementado utilizando o protocolo de comunicação *Peer-to-Peer*. Porém, para alcançar os objetivos desejados de flexibilidade e escalabilidade, percebeu-se a necessidade de alterar a forma de comunicação entre os componentes de sua arquitetura. Assim, o trabalho [39] implementou a Chamada de Procedimento Remoto (RPC). Esse modelo de comunicação permite realizar a comunicação de forma transparente, pois pode-se realizar a chamada de procedimentos que estão localizados em outras máquinas, sem que o usuário perceba. O *framework* utilizado para implementar o RPC será detalhado na Subseção 4.2.1.

Além disso, Moura et. al [39] implementaram uma política de armazenamento considerando a latência e o local a ser executado. Para auxiliar na organização e na coordenação do BioNimbuZ utilizaram um serviço voltado a sistemas distribuídos, chamado Apache ZooKeeper, que será mais detalhado na Subseção 4.2.2, e o protocolo SFTP para a transferência de arquivos de forma segura.

Após essa evolução, outros trabalhos continuaram o melhoramento do BioNimbuZ. Azevedo e Freitas [40] melhoraram a política de armazenamento, considerando novos parâmetros em sua política de compactação e transferência. Ramos [9] implementou uma interface gráfica que pode ser acessada por meio de um navegador *web*, antes o BioNimbuZ era acessado através do uso de linhas de comando. Além da interface, Ramos implementou o controlador de *jobs*, que faz a ligação entre a Camada de Interface e a Camada de Núcleo do BioNimbuZ. Recentemente, Moura [10] implementou um controlador de SLA, permitindo o gerenciamento de todo o ciclo de vida de cada SLA firmada entre um usuário e uma plataforma de nuvem federada. Vergara [38], implementou um controlador de elasticidade, permitindo o provisionamento de máquinas virtuais automático, além do gerenciamento dos recursos em cada máquina. Rosa [41], implementou o Serviço de Predição de Custos e Recursos, que agora informa os usuários sobre os custos financeiros e computacionais de maneira transparente. O trabalho [42], implementou o uso de *containers* nas execuções de *Workflows* de Bioinformática, de maneira que as máquinas virtuais são criadas já com os programas necessários para suas execuções.

4.2.1 Apache Avro

Apache Avro [43] é um sistema de serialização de dados e de RPC, desenvolvido pela Fundação Apache. Esse sistema possui diversas vantagens, como permitir a utilização de mais de um protocolo de rede e suportar mais de um formato de serialização. Se difere de seus concorrentes pela tipagem dinâmica, dados sem *overhead* e atribuição automática de ID's de campos. Assim como os demais produtos da Fundação Apache, o Avro é um software livre e amplamente utilizado. O Apache Avro suporta dados no formato binário

e JSON. Quanto aos seus protocolos de comunicação, pode-se utilizar o protocolo HTTP, assim como um protocolo próprio do Avro [44].

O Apache Avro suporta um grande volume de dados, e possui algumas características definidas pela própria Fundação Apache, como uma rica estrutura de dados com tipos primitivos, integração com diversas linguagens de programação e um formato de dados compacto, rápido e binário. Por esses motivos foi escolhido como *middleware* de Chamada Remota de Procedimento para o BioNimbuZ. Ele funciona de modo em que as requisições de execuções de procedimentos sejam serializadas para a execução remota, e ao receber a requisição, a máquina responsável pela execução recebe essa execução já de-serializada [44].

4.2.2 Apache Zookeeper

O Apache Zookeeper [45] é outro programa da Fundação Apache utilizado na coordenação de sistemas distribuídos. Ele utiliza um modelo de dados que simula uma estrutura de diretórios hierarquia, com a finalidade de facilitar a criação e a gestão de sistemas distribuídos, que podem ser de alta complexidade e difícil manutenção.

A organização hierarquia do Zookeeper utiliza espaços de nomes chamados de *znodes*. Cada *znode* tem a capacidade de armazenar até 1 Megabyte (MB) de informação, que são identificados pelo seu caminho na estrutura. Nos *znodes*, são armazenadas informações que facilitam o controle do sistema distribuído, tal como caminhos, dados de configuração, metadados e endereços. Esses dados são armazenados na memória para um acesso mais rápido [46].

Os *znodes* são classificados em dois tipos no ZooKeeper, os persistentes e os efêmeros. Essa classificação está relacionada ao seu armazenamento, os persistentes são mantidos mesmo após a queda de um provedor, sendo útil para armazenar informações que não se deseja perder, e os efêmeros são aqueles que podem ser apagados quando o cliente que o criou perde conexão por um determinado tempo. *Znodes* efêmeros no BioNimbuZ são criados para cada novo participante da federação, de forma que, caso um participante fique indisponível, o *znode* referente será eliminado e todos os componentes do sistema distribuído saberão que ele não se encontra disponível. Na Figura 4.2 pode ser visto a estrutura hierarquia dos *znodes* que foram implementados no BioNimbuZ.

Outro componente importante do ZooKeeper são os *watchers*, que funcionam como observadores de mudança, enviando alertas em cada alteração ocorrida em algum dos *znodes*. Este conceito é útil para sistemas distribuídos, pois permite o monitoramento constante do sistema, deixando que os *watchers* avisem quando um recurso estiver indisponível, por exemplo, ou quando um provedor estiver fora do ar. No BioNimbuZ os *watchers* são instanciados pelos serviços e controladores, por meio do *framework* Curator [47], o qual

também é fornecido pela Fundação Apache, para facilitar o CRUD (*Create Read Update Delete*) de *znodes*, e o instanciamento dos *watchers* para os serviços do ZooKeeper.

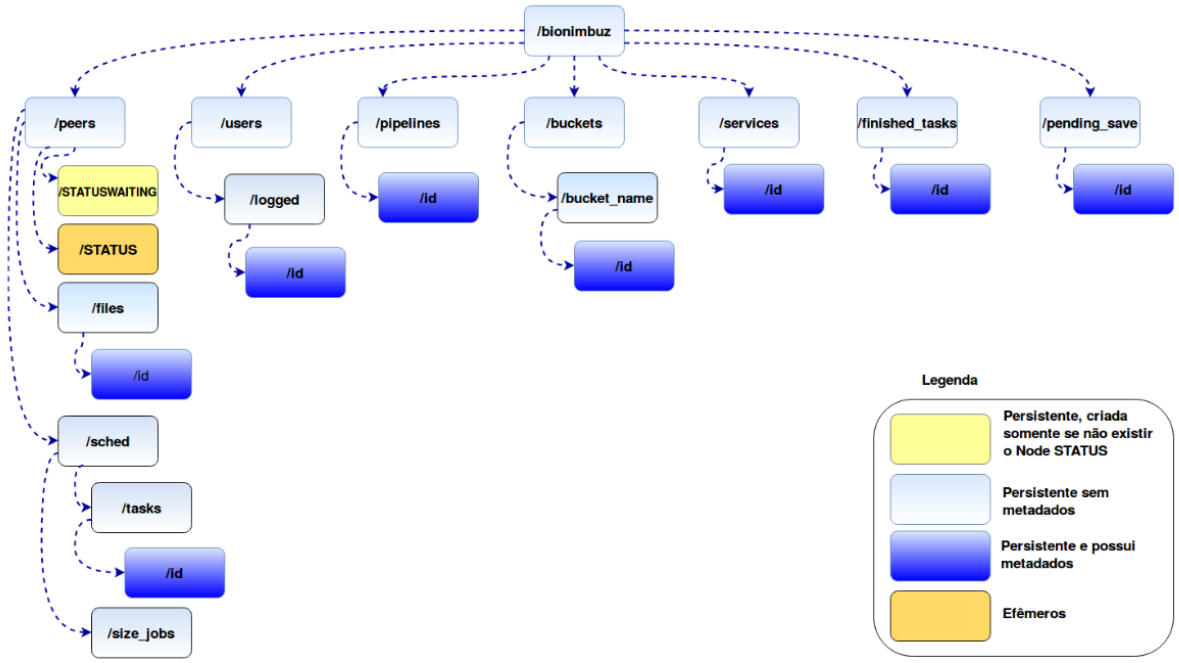

Figura 4.2: Estrutura Hierárquica dos Znodes no BioNimbuZ [9].

4.3 Arquitetura do BioNimbuZ

A arquitetura do BioNimbuZ é organizada de forma hierárquica e distribuída, conforme apresentado na Figura 4.3. Ela representa a interação entre as quatro camadas: Aplicação, Integração, Núcleo e Infraestrutura.

A Camada de Aplicação é responsável pela interação com o usuário. A Camada de Integração é responsável pela transmissão de dados entre as Camadas de Aplicação e a de Núcleo. A terceira camada, Camada de Núcleo, é responsável por realizar toda a gerência da plataforma e seus serviços, e por se comunicar com a quarta camada. A Camada de Infraestrutura é responsável pelos *plugins* de cada provedor, estes, mapeiam as requisições e os enviam a para cada provedor. A seguir serão detalhadas as características e o funcionamento de cada uma dessas quatro camadas.

4.3.1 Camada de Aplicação

A Camada de Aplicação é responsável pela integração de toda a plataforma com o usuário. Por meio desta camada os usuários podem fazer *log-in* no sistema BioNimbuZ, acessando

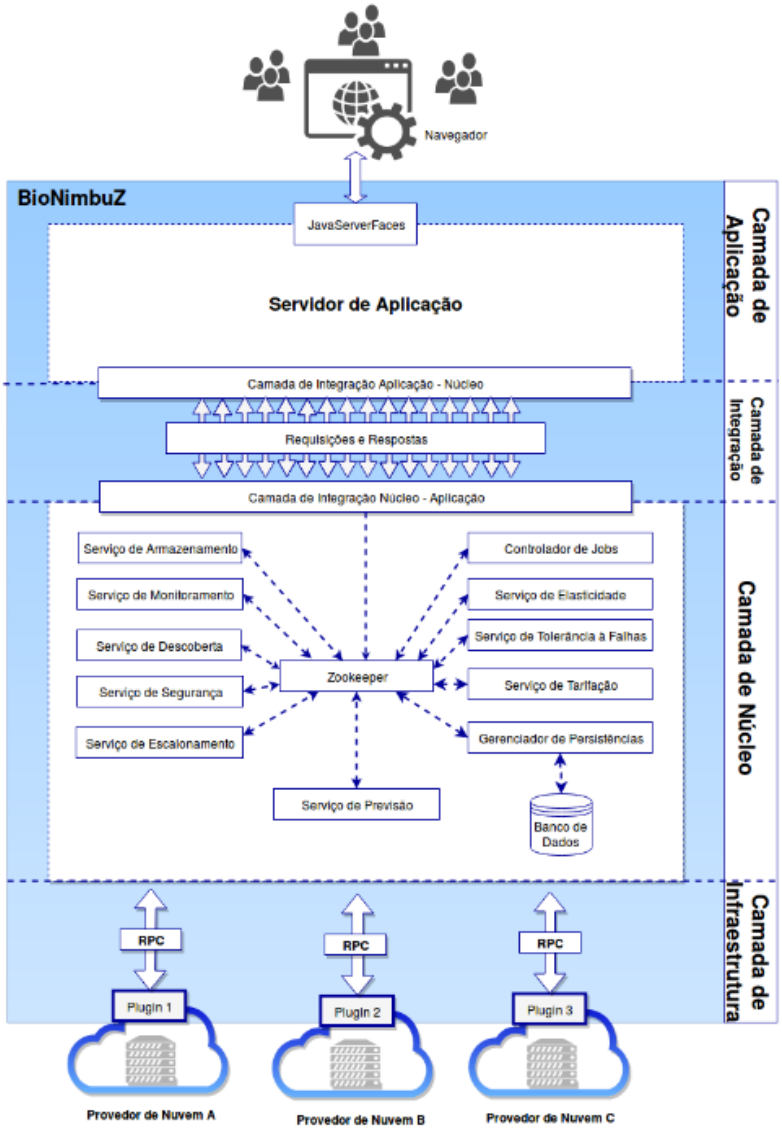

Figura 4.3: Arquitetura da Plataforma BioNimbuZ [10].

uma página *web*. Após o *log-in* na aplicação, o usuário se depara com a página inicial, conforme demonstrado na Figura 4.4, na qual há um sistema gerenciador de *workflows* científicos.

Na página inicial, o sistema apresenta algumas opções relacionadas ao gerenciamento dos *workflows* e gerenciamento dos arquivos utilizados nestes *workflows*. Assim, um usuário pode seguir para a criação de um novo *workflow*, principal página da aplicação, ou verificar os *logs* e os estados dos *workflows* já executados. Na área de gerenciamento de arquivos o usuário pode incluir novos arquivos, listar os arquivos já armazenados, excluí-los e realizar o seu *download*.

Por meio da interface web os usuários podem criar *workflows* de maneira gráfica,

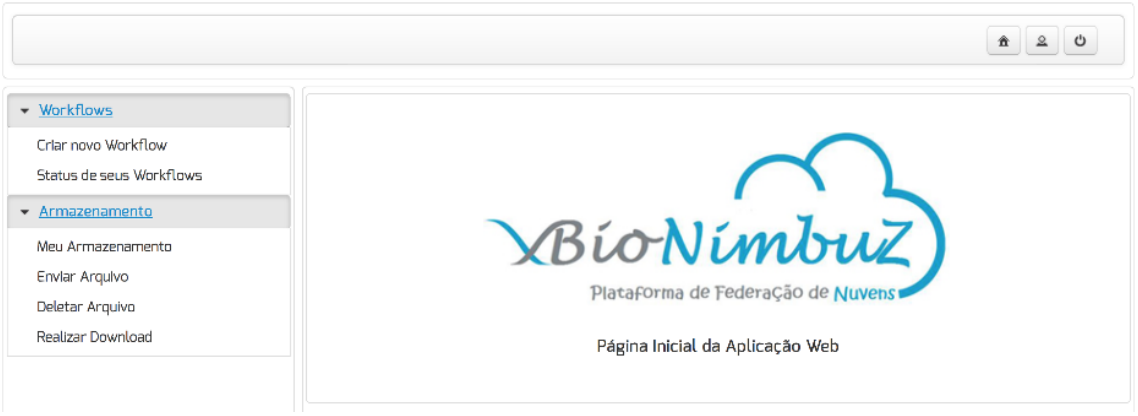

Figura 4.4: Página Inicial da Aplicação *Web* do BioNimbuZ [9].

ligando passos, indicando dependências, incluindo parâmetros e arquivos que serão utilizados conforme pode ser visto na Figura 4.5.

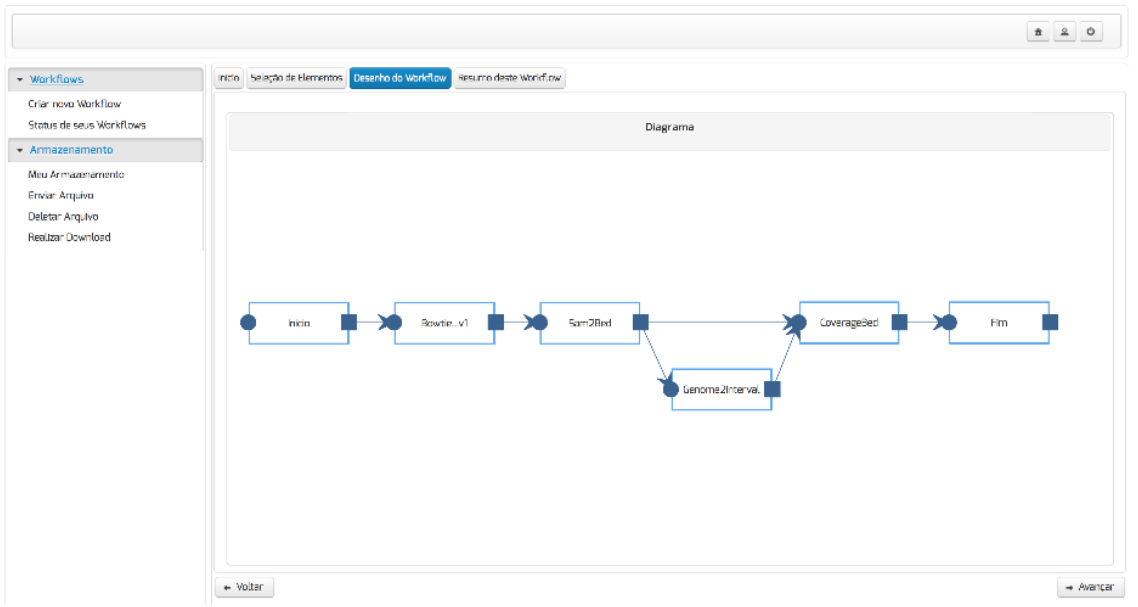

Figura 4.5: Tela de Montagem de Fluxo do Workflow da Aplicação [9].

4.3.2 Camada de Integração

A Camada de Integração é responsável por integrar a Camada de Aplicação e a Camada de Núcleo. Essa integração é feita utilizando mensagens entre essas camadas, por meio de *webservices*. Assim, a integração de camadas permite que a Camada de Aplicação requisite tarefas ao núcleo, e receba respostas da Camada de Núcleo, enviando essas respostas para a Camada de Aplicação [37].

As mensagens enviadas por meio de *webservices* utilizam REST (*REpresentation State Transfer*). O REST é voltado para sistemas baseados na Internet e tem sido amplamente utilizado na integração de sistemas, por utilizar operações do protocolo HTTP, tais como PUT, GET e DELETE. Assim, a interface desenvolvida na camada de núcleo é independente da aplicação que irá acessá-lo [9].

4.3.3 Camada de Núcleo

A Camada de Núcleo é responsável por todo o gerenciamento de tarefas no ambiente de nuvens federadas. Essas tarefas incluem o descobrimento de novos provedores e recursos, o controle de acesso ao usuários, o agendamento de tarefas, o gerenciamento de *jobs* e o controle dos arquivos armazenados na federação. Todas essas tarefas são distribuídas em diversos módulos, chamados de serviços, ou alguns de controladores. A fim de exercerem suas funcionalidades, os serviços e os controladores interagem entre si por meio do gerenciamento de troca de mensagens provido pelo ZooKeeper. A seguir serão descritos os serviços e os controladores que contemplam a Camada de Núcleo do BioNimbuZ:

• **Controlador de Jobs:** Sua responsabilidade é gerenciar os pedidos da Camada de Aplicação e garantir que os resultados sejam entregues de forma correta para cada uma das requisições, e mantê-los para que possam ser consultados posteriormente. Recebe do usuário a requisição da Camada de Aplicação e faz a ligação com o núcleo do BioNimbuZ, verificando as credenciais dos usuários por meio do Serviço de Segurança;

• **Controlador de SLA:** É responsável por todo o gerenciamento do ciclo de vida do SLA. O usuário define o SLA desejada ao submeter uma tarefa no BioNimbuZ, e o controlador de SLA acompanha se os parâmetros acordados estão sendo cumpridos até a finalização da tarefa. No caso do SLA não ser cumprido, o controlador aplica multas ao provedor que disponibiliza créditos para o usuário;

• **Serviço de Elasticidade:** É o serviço responsável pela elasticidade da nuvem tanto vertical quanto horizontal. Em um caso de elasticidade vertical, o serviço verifica se há falta de recursos, caso seja necessário, são incrementados os recursos, como número de CPUs, quantidade de memória, entre outros. Assim como pode ser feito o provisionamento de novos recursos, em caso de ociosidade, os recursos podem ser desalocados. Também é de responsabilidade deste serviço a elasticidade horizontal, criando novas VMs se houver necessidade, assim como excluindo as ociosas, para um melhor gerenciamento de recursos na federação;

• **Serviço de Monitoramento:** É o responsável pelo acompanhamento dos recursos da federação, junto ao ZooKeeper, com a utilização de *watchers* e tratando os alertas enviados pelos mesmos. Também permite a recuperação dos dados principais utilizados

pelos módulos de cada servidor BioNimbuZ, armazenados na estrutura do ZooKeeper, para possíveis reconstruções ou garantias de execuções de serviços solicitados;

• **Serviço de Descobrimento:** É o serviço que identifica e mantém informações a respeito dos provedores de nuvens da federação, tais como processamento, latência de rede e capacidade de armazenamento, além de armazenar detalhes sobres os parâmetros de execução de arquivos de entrada e de saída. Todas as informações relevantes dos provedores são armazenadas no ZooKeeper em *znodes* [45]. Conforme os provedores têm suas informações modificadas, os *watchers* informam os demais serviços, assim mantendo todos sempre atualizados;

• **Serviço de Escalonamento:** Sua responsabilidade é distribuir os *jobs* recebidos em instâncias menores, chamadas de *tasks*, e distribuir essas *tasks* entre os provedores selecionados. Esse serviço também é responsável pelo acompanhamento de todo o ciclo de vida do *job*, mantendo os registros das execuções já escalonadas. Para realizar a distribuição de tarefas na federação, os escalonador utiliza métricas como a latência de rede, balanceamento de carga, tempo de espera, capacidade de processamento, entre outras, visando atender o que lhe foi acordado no SLA. Atualmente, o BioNimbuZ implementa três diferentes políticas de escalonamento, cada uma visando um objetivo [48][49] [36];

• **Serviço de Armazenamento:** É o responsável pela política de armazenamento dos arquivos que são utilizados ou mantidos pelo BioNimbuZ. A partir de informações providas pelo serviço de descobrimento, são analisados parâmetros para eleger a melhor opção de armazenamento. Esses parâmetros incluem, latência de rede, localização física dos provedores e capacidade de armazenamento. É implementada a política de replicação, visando a diminuição de custos. Inicialmente o serviço de aprimoramento foi desenvolvido por Moura [50] e, posteriormente, melhorado por [40] [51];

• **Serviço de Tarifação:** Seu objetivo é calcular o valor a ser pago pelo usuário devido a sua utilização na plataforma. Isso é possível por meio da comunicação com o Serviço de Monitoramento para verificar o tempo de execução das tarefas e a quantidade de máquinas alocadas por estas. Assim, é possível verificar quais recursos foram alocados para determinada carga de trabalho submetida ao usuário, calculando, dessa forma, o preço devido por cada usuário da plataforma;

• **Serviço de Segurança:** Segurança em um ambiente de nuvens é um requisito essencial e extremamente importante. No BioNimbuZ esse serviço é responsável por diversas tarefas, uma delas é a autenticação dos usuários, que verifica se o usuário é quem ele realmente diz ser. Após os usuários serem autenticados, suas ações são controladas por autorização, de forma que ele só pode realizar as ações em que possui permissão. O uso de criptografia na comunicação é utilizada para garantir o sigilo dos dados. Também é implementada a verificação de integridade dos arquivos, de forma que estes não possam

ser modificados por fatores externos a federação;

• **Serviço de Predição:** É responsável por prover informações aos usuários, no auxílio da escolha de um ambiente computacional que execute seu *workflow* científico de forma transparente, com estimativa de tempo, custos financeiros e recursos a serem utilizados. Ele funciona de forma que o usuáriopossa preencher as informações a respeito do *workflow*, processam os dados e enviam um *feedback* ao usuário, estimando o tempo, o custo e os melhores recursos disponíveis.

Devido a importância do serviço de tolerância a falhas neste estudo, o mesmo será detalhado na Seção 4.4.

4.3.4 Camada de Infraestrutura

O principal objetivo da Camada de Infraestrutura é prover uma interface de comunicação entre os provedores de nuvem e o BioNimbuZ, utilizando *plugins* para isso. Esses *plugins* mapeiam as requisições provenientes da Camada de Núcleo para os comandos específicos de cada provedor, individualmente. Esta camada consiste em todos os recursos que os provedores de nuvens colocam a disposição da federação, e os seus respectivos *plugins* de integração. Assim sendo, faz-se necessário desenvolver um *plugin* para cada plataforma de nuvem, tais como o *Amazon*, *Azure*, *Google* e *Digital Ocean*, entre outros.

4.4 Serviço de Tolerância a Falhas

Este serviço garante que todos os serviços do núcleo estejam sempre disponíveis. Em um ambiente de nuvens, falhas de máquinas ocorrem, e é conhecido por toda a comunidade que essas falhas fazem parte de um processo normal. Assim, qualquer nuvem federada deve ser desenvolvida para recuperação de falhas e alta disponibilidade do sistema. Logo, um serviço de tolerância a falhas é parte essencial do BioNimbuZ, e seu objetivo é prover alta disponibilidade e resiliência contra falhas periódicas e falhas transientes.

Por se tratar de uma plataforma distribuída, o serviço de tolerância a falhas, também deve atuar de tal forma, ou seja, estar presente em outros serviços, monitorando seus estados. Por meio da organização do ZooKeeper, toda a disponibilidade dos recursos e dos serviços é monitorada pelos *watchers*, em caso de alguma falha, estes imediatamente enviam um alerta sobre a indisponibilidade.

Além da tolerância a falhas em seu núcleo, implementada pelo ZooKeeper, o BioNimbuZ utiliza outras técnicas, como replicação, migração e balanceamento de carga, descritas nas subseções 4.4.1, 4.4.2 e 4.4.3 respectivamente.

4.4.1 Replicação

A replicação de dados está presente no Serviço de Armazenamento. O BioNimbuZ mantém, pelo menos, mais duas cópias de cada arquivo distribuído em suas máquinas virtuais, e no caso de algum alerta disparado por um *watcher*, a recuperação do arquivo é iniciada nas outras máquinas disponíveis.

Santos [52] implementou um novo modelo de armazenamento no qual os provedores de nuvem utilizados já realizam a replicação internamente, não sendo necessária o uso dessa técnica pela camada de núcleo do BioNimbuZ.

A replicação de dados também é realizada no ZooKeeper, que está presente em diversos serviços. O ZooKeeper possui um algoritmo para a eleição de um líder, e os demais são chamados de seguidores. Em caso de indisponibilidade do líder o algoritmo de eleição é executado novamente e um novo líder é escolhido, conforme é mostrado na Figura 4.6.

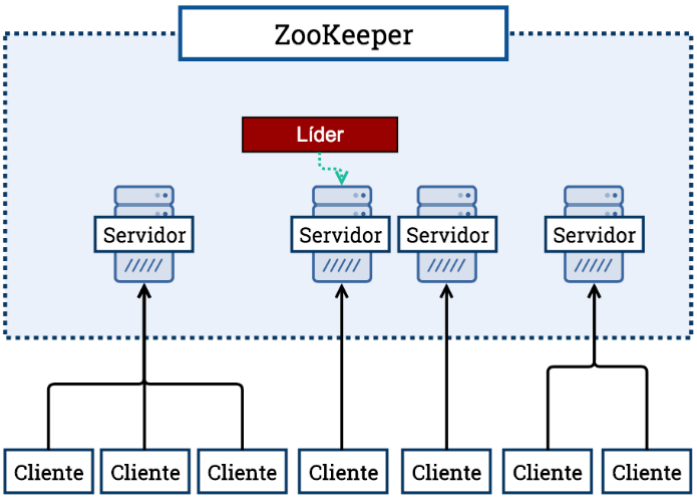

Figura 4.6: Modelo de Serviço Utilizado no ZooKeeper [9].

4.4.2 Migração

A migração está presente no Serviço de Escalonamento. Quando é recebido um alerta de indisponibilidade de um determinado recurso, todas as tarefas presentes nesse recurso, que ainda não executaram ou ainda estão em execução, são re-escalonadas novamente, agora para outras máquinas da federação. Essa migração é possível através dos *watchers*, que disparam alertas assim que uma indisponibilidade é detectada, avisando sobre os problemas de algum recursos na federação.

4.4.3 Balanceamento de Carga

O balanceamento de carga está presento no Serviço de Escalonamento. Nos algoritmo de escalonamento utilizados pelo BioNimbuZ, o balanceamento de carga é uma das métricas utilizadas para seleção das máquinas que serão executadas as tarefas. No caso de uma máquina está sobrecarregada de tarefas, a mesma pode apresentar lentidão, estouro de recursos, e até no pior dos casos um defeito de travamento, assim, é importante distribuir as tarefas de maneira igualitária, visando uma melhor utilização dos recursos.

4.5 Considerações Finais

Neste capítulo foram apresentadas as principais características da plataforma BioNimbuZ, assim como sua arquitetura, os serviços e o conceito de *workflow* científico. Visto a importância do serviço de tolerância a falhas, e que atualmente o BioNimbuZ não implementa a técnica de *checkpoint* em nenhum de seus serviços, este trabalho propõe o uso dessa técnica na execução dos *workflows*, conforme apresentado no Capítulo 5.

Capítulo 5

Checkpoint-Restart no BioNimbuZ

Neste capítulo será apresentada a proposta do uso de *checkpoint* para o ambiente de nuvens federadas, utilizando a plataforma de nuvens federadas BioNimBuZ, como estudo de caso. Para isso, inicialmente, na Seção 5.1 será apresentada a visão geral da técnica proposta. Na Seção 5.2 será definida e detalhada a arquitetura do serviço proposto. Na Seção 5.3 será apresentado o estudo de caso, ambiente e *workflow* utilizados. Na Seção 5.4 serão mostrados os resultados obtidos com a implementação do serviço. Na Seção 5.5 serão apresentadas as considerações finais a respeito deste capítulo.

5.1 Serviço Proposto

O objetivo do serviço proposto neste trabalho é que ele seja capaz de reduzir o tempo perdido na execução de *workflows* de Bioinformática, nos casos de falhas transientes, atuando de maneira automática no reenvio de tarefas utilizando a técnica de *checkpoint*.

Workflows de bioinformática podem exigir um longo tempo de processamento, durante esse período, falhas em algum componente podem causar a interrupção inesperada do processo em execução. Devido a isso, foi verificada a necessidade de implementar novas técnicas de tolerância a falhas na plataforma de federação BioNimbuZ como o reenvio de tarefas para execução. Visto que o tempo de processamento de cada tarefa é longo, e o reenvio de tarefas ocasiona perda de todo o processamento já realizado, a técnica de *checkpoint* também foi implementada para reduzir o tempo de reexecução.

No BioNimbuZ cada tarefa passa por um ciclo de estados, na Figura 5.1, é demonstrado o fluxo de estados de um *workflow* de três tarefas finalizado com sucesso. Em seguida são descritos os possíveis estados de uma tarefa.

- **Pendente:** A tarefa está pendente, esperando para ser escalonada;
- **Esperando:** A tarefa já foi escalonada e aguarda a execução;
- **Em execução:** A tarefa está em execução no momento;

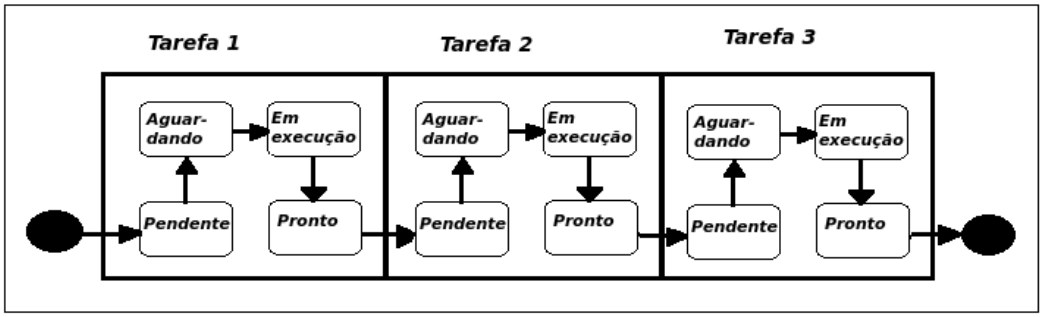

Figura 5.1: Fluxo de Estados das Tarefas de um *Workflow* Finalizado com Sucesso no BioNimbuZ.

- **Pronta:** A tarefa foi finalizada com sucesso;
- **Cancelada:** A tarefa foi cancelada pelo usuário;
- **Erro:** A tarefa finalizou com erros.

Conforme visto na Seção 4.4, o BioNimbuZ implementa algumas técnicas de tolerância a falhas, todavia,ele não faz uso da técnica de *checkpoint*. Assim, neste trabalho o serviço de tolerância a falhas do BioNimbuZ será incrementado com duas técnicas, reenvio de tarefas e *checkpoint* de tarefas.

No caso de uma falha em alguma das tarefas de um *workflow*, o *workflow* é imediatamente finalizado com erros. Assim, todo o processamento realizado até o momento da falha é perdido, e as futuras tarefas não são executadas, causando a perda de todo o*workflow*. No exemplo da Figura 5.2 é mostrado um erro na execução da segunda tarefa.

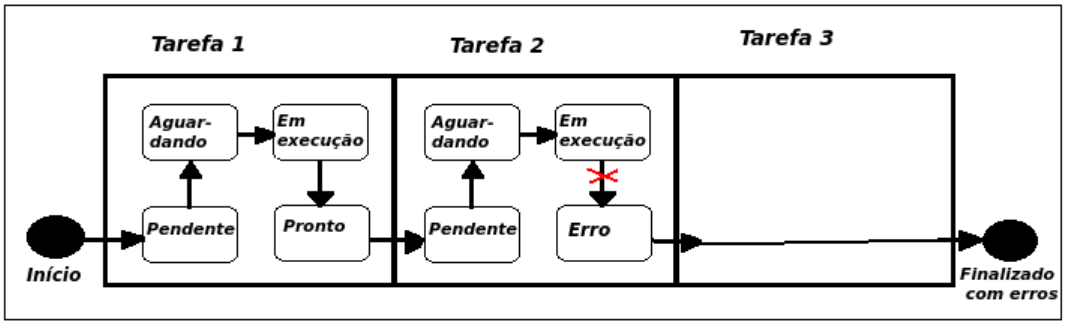

Figura 5.2: Fluxo de um *Workflow* Finalizado com Erros no BioNimbuZ.

Assim sendo, o trabalho proposto implementa o reenvio de tarefas para execução no mesmo recurso, visando realizar mais uma tentativa de execução antes entender como perdido todo o *workflow*, conforme demonstrado na Figura 5.3.

Desta forma, o fluxo da Figura 5.3 funciona da seguinte maneira, após detectado que uma tarefa terminou devido a uma finalização inesperada do processo que a estava

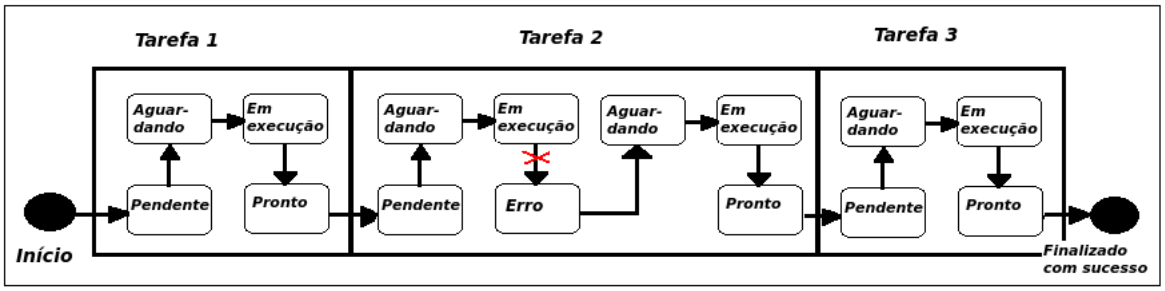

Figura 5.3: Fluxo de um *Workflow* com Reenvio de Tarefas no BioNimbuZ.

executando, o estado dessa tarefa é modificado para "esperando", assim, a tarefa volta para a fila de execução e é novamente executada pelo recurso no qual foi escalonada. No caso da implementação com *checkpoint*, a tarefa volta a ser executada a partir do seu último estado válido salvo, dessa forma, o tempo perdido é reduzido. Isso ocorre porque a tarefa re-iniciará para executar apenas as instruções a partir do último *checkpoint* redigido, pois todo o seu estado anterior estará a salvo.

5.2 Arquitetura

A ferramenta escolhida para realizar o *checkpoint* dos processos foi o DMTCP (*Distributed MultiThreaded Checkpointing*). O DMTCP realiza *checkpoints* em nível de usuário em sistemas distribuídos e não distribuídos, sem modificar o código da aplicação ou o código do sistema operacional. Outras ferramentas como o *Berkeley Lab Checkpoint/Restart* (BLCR) foram analisadas, mas devido a sua implementação em nível de sistema, não foi considerada a opção ideal.

O DMTCP foi escolhido devido a maturidade do seu projeto, que possui mais de 12 anos, o vasto número de aplicações Linux suportadas, por realizar o *checkpoint* de aplicações *multithreads* e por suportar migração de processos em ambientes heterogêneos, uma característica essencial para um ambiente de nuvens federadas. Foi desenvolvido nas linguagens C/C++ e possui licença pública GNU.

O DMTCP possui um processo chamado de coordenador, que é responsável pelo gerenciamento dos *checkpoints*. Nele, as aplicações se conectam e enviam mensagens a respeito do seu estado, assim, os estados são salvos em disco e estão disponíveis para possíveis reexecuções. O coordenador pode estar presente na mesma máquina que a aplicação ou em uma máquina diferente. A Figura 5.4 exemplifica o funcionamento do coordenador.

A integração do DMTCP com o BioNimbuZ é feita na chamada das aplicações, nos comandos de sistema operacional, onde antes do comando da aplicação é chamado o comando dmtcp-lauch, que fica responsável por enviar as informações do processo da apli-

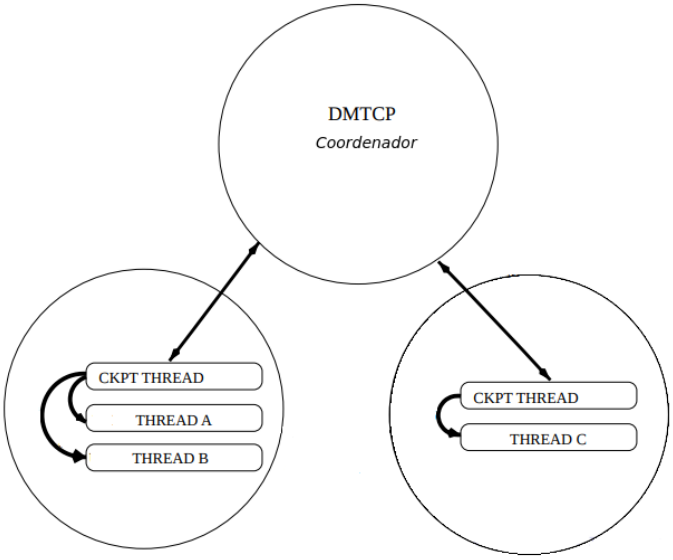

Figura 5.4: Arquitetura do Coordenador de *Checkpoint*.

cação ao coordenador. O dmtcp-launch também é responsável por pausar as aplicações para salvar suas informações no arquivo de *checkpoint*, de maneira que um novo arquivo é criado e posteriormente substitui o anterior. Na imagem do *checkpoint* são salvas informações a respeito do processo como as variáveis contidas na memória RAM, assim como as bibliotecas utilizadas. O dmtcp implementa verificação de soma para integridade da imagem de *checkpoint*.

Na implementação realizada, o BioNimbuZ detecta quando o processo da aplicação foi encerrado inesperadamente, assim, iniciando o processo de *checkpoint*, de maneira que o script de *restart* é iniciado, lendo as informações contidas no arquivo de *checkpoint* e restaurando as informações do processo no sistema operacional.

O DMTCP permite a personalização de diversos parâmetros para se adequar as necessidades de cada implementação. A seguir serão demonstrados os principais comandos, para que servem e alguns parâmetros relevantes:

- **dmtcp-coordinator:** Inicia o processo coordenador, que recebe conexões dos processos em que são realizados o *checkpoint*. Nele podem ser adicionados alguns parâmetros relativos ao tempo de intervalo, uso de compactação dos arquivos de *checkpoint* e porta a ser utilizada para receber conexões das aplicações;
- **dmtcp-command:** Comando utilizado para interagir com o coordenador, assim, podem ser verificados os estados dos processos, o tempo de intervalo e listar os *nodes* conectados. Também é possível enviar comandos para realizar *checkpoints* eventuais, matar os processos e encerrar o coordenador;
- **dmtcp-restart:** Comando utilizado para reiniciar um processo a partir do seu último *checkpoint*. Nesse comando pode ser passado, como parâmetro, um arquivo de

checkpoint, caso nenhum parâmetro seja utilizado, o *restart* será realizado a partir do último arquivo de *checkpoint*.

No BioNimbuZ, o serviço de *Checkpoint* está diretamente relacionado com o serviço de escalonamento, conforme demonstrado na Figura 5.5. O código do *plugin* de execução de tarefas foi modificado, permitindo chamadas da classe que implementa os métodos e as variáveis responsáveis pelo DMTCP.

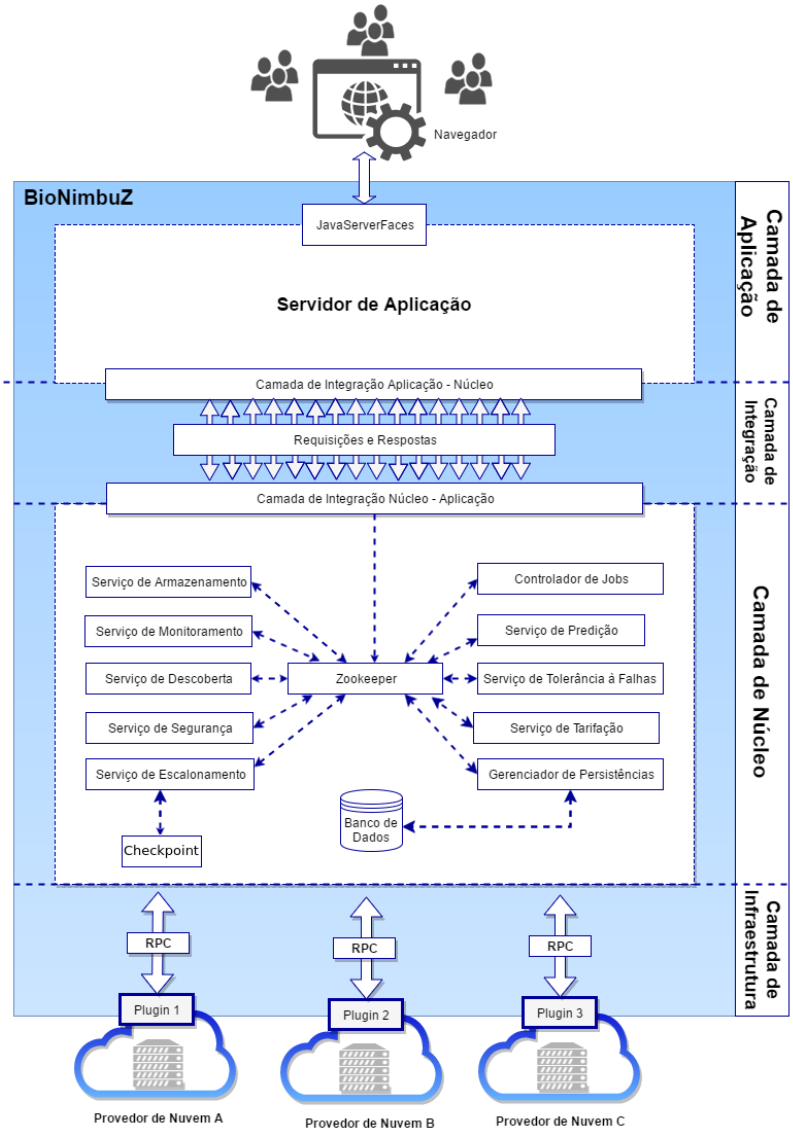

Figura 5.5: Arquitetura do BioNimbuZ com *Checkpoint*.

5.3 Estudo de Caso

O estudo de caso tem como objetivo testar o serviço de *checkpoint* proposto, e buscar tornar o BioNimbuZ uma plataforma mais tolerante a falhas, agindo de maneira rápida

Tabela 5.1: Informações Técnicas da Máquina.

Característica	**Descrição**
Marca	Samsung
Processador	Intel Core i7
Num. Núcleos	4
Memória RAM	12GBs
Disco	SSD 500 GBs

na execução de procedimentos para minimizar o tempo perdido na falha de tarefas.

Para os testes foi utilizado um *workflow* de Bioinformática composto por seis tarefas. Essas tarefas são a Filtragem, a Construção de índices para mapeamento, o Mapeamento, a Conversão de formato, a Ordenação e, por fim, a Contagem, sendo executados pelos programas *sickle* [53], *hisat2-build* [54], *hisat2* [54], *samtools view* [55], *samtools sort* [55] e *htseq-count* [56] respectivamente.

Devido ao fato do uso do *checkpoint* ser melhor aproveitado em tarefas de longa duração, as tarefas foram agrupadas em 3 etapas, sendo a primeira composta pela Filtragem, Criação de índices e Mapeamento; a segunda etapa pela Conversão de Formatos e Ordenação, e a última etapa responsável pela Contagem. Na Figura 5.6 é demonstrado o fluxo de execução criado a partir da interface *web* do BioNimbuZ.

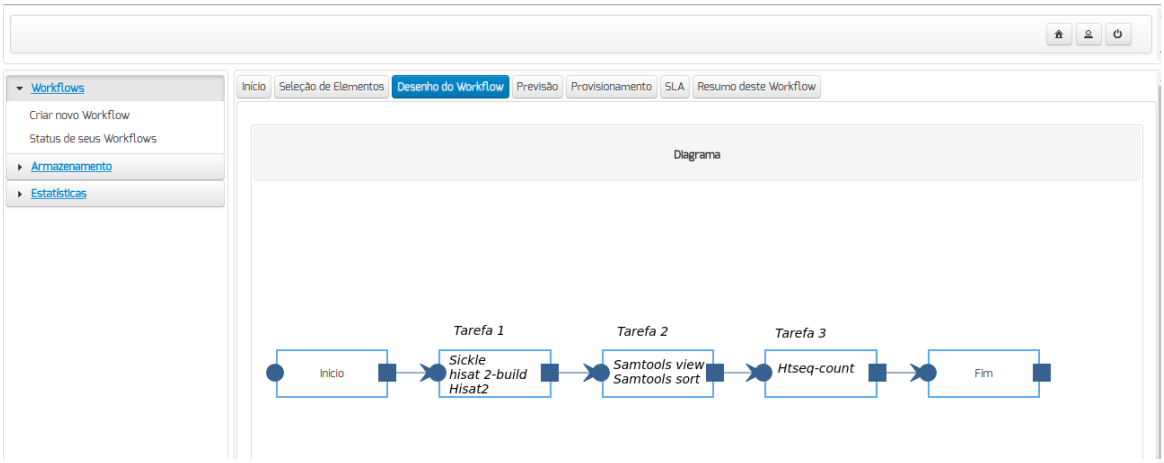

Figura 5.6: Fluxo do *Workflow* no BioNimbuZ.

Devido as técnicas utilizadas não implementarem a migração de tarefas em diferentes recursos computacionais, os testes foram realizados em um computador pessoal, sendo este o suficiente para avaliar o funcionamento do serviço. As características do computador utilizado estão descritas na tabela 5.1. Na Figura 5.7 pode ser vista a etapa de provisionamento de recursos no BioNimbuZ.

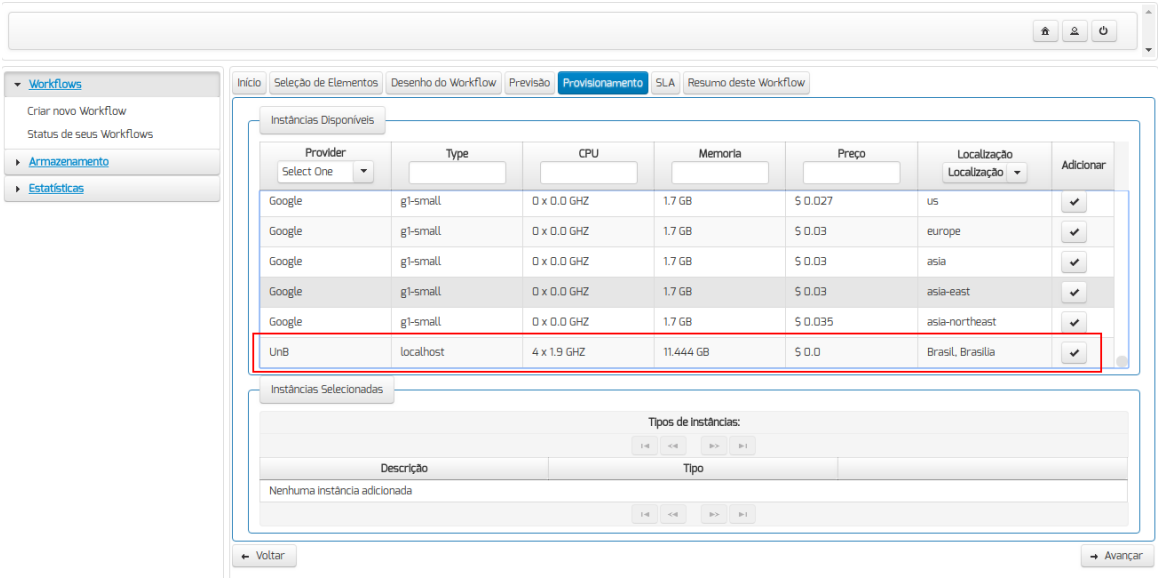

Figura 5.7: Etapa de Provisionamento de Recursos no BioNimBuZ.

5.4 Testes e Resultados Obtidos

Para analisar a eficiência e a funcionalidade do serviço proposto neste trabalho, os testes foram definidos em dois grupos. Os testes de execução de *workflow* sem falhas e os testes com falhas. O primeiro deles foram os testes sem falhas, nos quais as tarefas eram executadas completamente para avaliar o tempo de *overhead* do *checkpoint*. O segundo grupo de testes realizados foram os com falhas, no qual é avaliado a comparação dos resultados entre execuções com reenvio de tarefas e execuções com uso de *checkpoint*. Na tabela 5.2 são apresentados os valores em *Megabytes* do tamanho dos arquivos de imagem do *checkpoint* para cada programa executado.

5.4.1 Overhead do Checkpoint

O *overhead* é a principal desvantagem da implementação da técnica de *checkpoint*. No momento em que o serviço realiza o *checkpoint*, o processo é pausado para que suas informações sejam salvas em disco, após serem salvos, o processo continua sua execução, assim, essas pausas para escrita aumentam o tempo de execução total de cada tarefa. Dois fatores são relevantes no tempo de *overhead*, primeiro, o intervalo de *checkpoint*, quanto menor, mais frequente o processo será pausado, assim, maior será o *overhead*. O segundo fator são as variáveis carregadas em memória pela aplicação, programas que armazenam uma grande quantidade de dados em memória demoram mais tempo para que esses dados sejam escritos em disco.

Tabela 5.2: Tamanho da Imagem de *Checkpoint*.

Programa	**Tamanho em Megabytes**
Sickle	6,57
Hisat2-build	0,05
Hisat2	4,23
Samtools view	3,5
Samtools sort	912
Htseq-count	930

Os testes foram realizados em cada tarefa separadamente, visando avaliar o desempenho do uso do *checkpoint* em diferentes aplicações. Para cada tarefa foi calculado o seu tempo de execução total, sem falhas, diversas vezes. Foram realizadas vinte execuções completas, sendo que dez não implementaram a técnica de *checkpoint* e a outra metade utilizou o *checkpoint*. A média aritmética da execução pode ser vista na Figura 5.8.

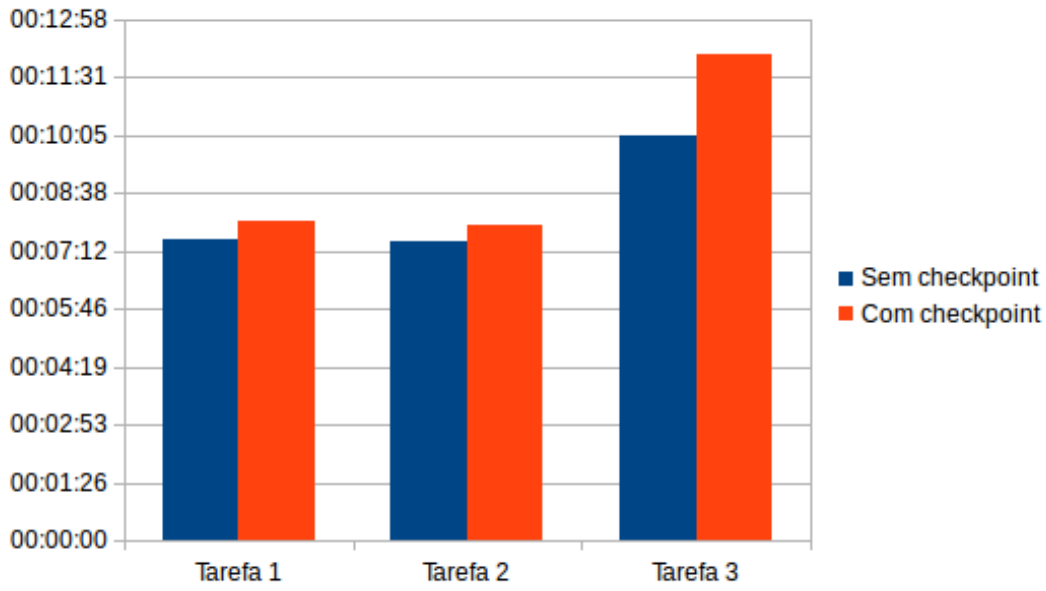

Figura 5.8: Análise do *Overhead* com a Técnica de *Checkpoint*.

Para o cálculo do *overhead*, foi obtido o tempo médio de execução de cada tarefa sem *checkpoint* e com o *checkpoint*. O valor médio das tarefas com *checkpoint* foi dividido pelo valor médio das tarefas sem *checkpoint*, resultando em um *overhead* médio de 11,56

5.4.2 Comparação de Desempenho da Técnica de *Checkpoint*

A principal vantagem da implementação da técnica de *checkpoint* é o aproveitamento do processo até o seu último estado válido. Enquanto o reenvio de tarefa apresenta a perda de todo o processamento anterior a uma falha, o mesmo não ocorre no *checkpoint*.

Novamente os testes foram realizados em cada tarefa separadamente, visando avaliar o desempenho em cada aplicação. Os erros foram inseridos logo após cada intervalo de *checkpoint*. O intervalo definido para os testes foi de dois minutos, devido a uma análise de melhor tempo de intervalo, realizada previamente na execução das tarefas. Após a inserção do erro, o BioNimbuZ imediatamente enviava a tarefa para a fila de execução. Ao serem processadas as tarefas eram executadas de acordo com o seu método de tolerância a falhas.

Durante estes testes, foi verificado que o tempo total de execução de tarefas com o uso do *checkpoint* não é alterado em relação ao instante da falha, visto que nos testes executados o tempo de execução é totalmente reaproveitado. No caos do reenvio de tarefas, o tempo total é o tempo anterior a falha, somado ao tempo total da tarefa. Assim, quanto maior o tempo de execução antes de uma falha, melhor será o desempenho do *checkpoint* em relação ao reenvio de tarefas, conforme apresentado nas Figura 5.9, 5.10 e 5.11.

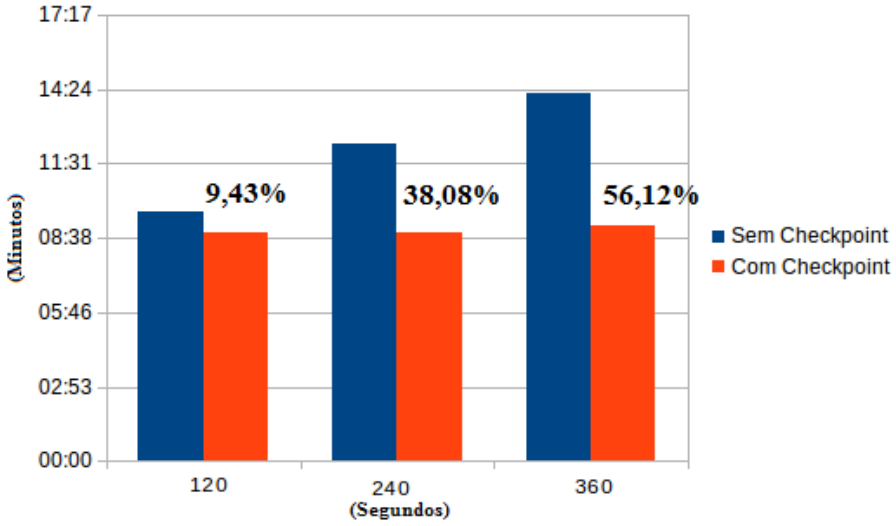

Figura 5.9: Comparação da Tarefa 1.

5.5 Considerações Finais

Neste capítulo foi apresentado o serviço de *checkpoint* proposto, seus objetivos e o seu funcionamento. Para isso, foi demonstrado o estudo de caso, os testes relacionados as suas vantagens e desvantagens e os resultados de desempenho da técnica de *checkpoint* proposta para o BioNimbuZ

Além de avaliar qual é o impactado da adição do serviço de *checkpoint* ao BioNimbuZ, também foi possível observar nos testes, a redução do tempo de execução de tarefas

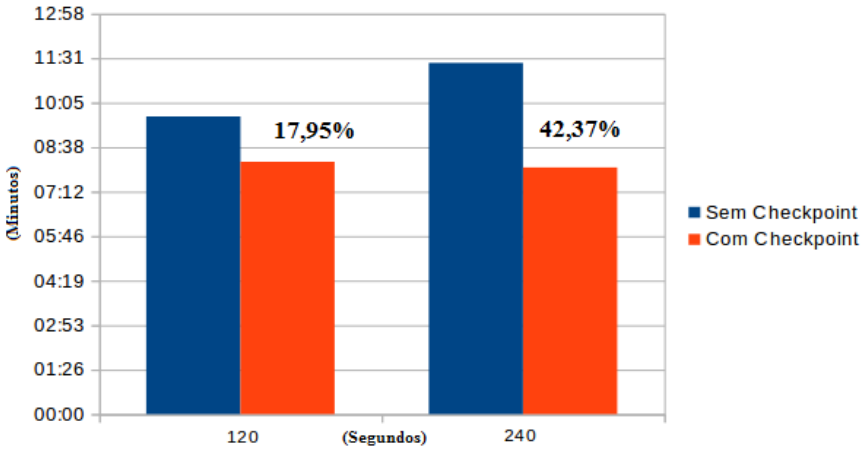

Figura 5.10: Comparação da Tarefa 2.

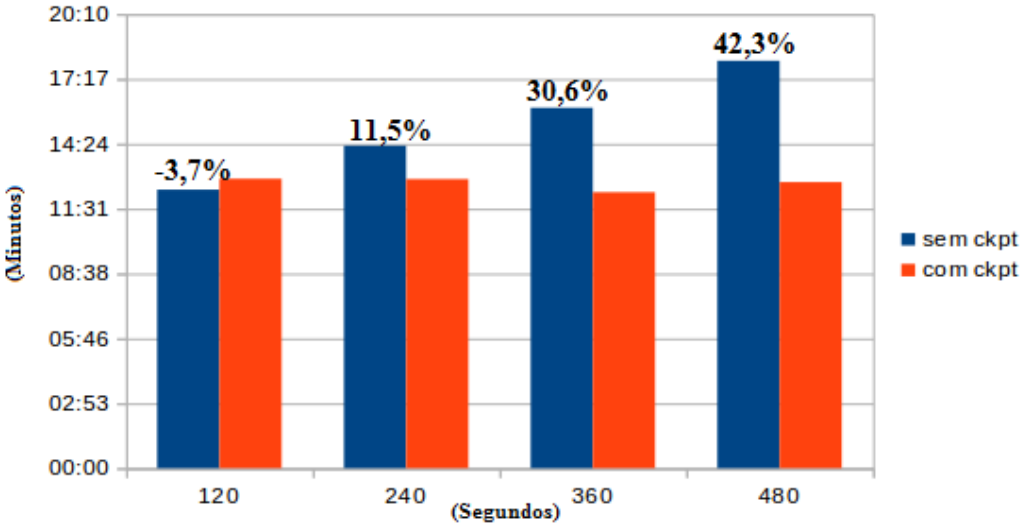

Figura 5.11: Comparação da Tarefa 3.

que o serviço proporcionou, permitindo, assim, ações imediatas de tolerância a falhas na plataforma.

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho foi proposta uma nova implementação de técnica de tolerância a falhas para a plataforma de federação de nuvens BioNimbuZ, de modo a ampliar a sua resiliência e aprimorar o seu serviço de tolerância a falhas. A técnica utilizada foi o *checkpoint-restart*, que foi implementado por meio do software livre DTMCP.

O uso da técnica de *checkpoint* reduz o tempo de reinicialização de aplicações de Bioinformática e, deste modo, resulta em um mais curto tempo de provisionamento de recursos, assim como, menor gasto financeiro e inferior tempo de execução das aplicações em caso de falhas. Para isto, o fluxo de tarefas do BioNimbuZ foi modificado, de modo que é realizada uma nova reexecução a partir do seu último estado válido. Assim, a técnica implementada garante que as aplicações possam ser reexecutadas sem perder o trabalho já realizado, sendo de suma importância para tarefas de longa duração.

Testes que executaram um *workflow* Bioinformática, comprovaram o desempenho superior do *checkpoint* em relação a técnica de reenvio de tarefas em caso de falhas. Também foram realizados testes para medir os efeitos colaterais da implementação do *checkpoint*.

Como trabalhos futuros, sugere-se a integração do *checkpoint* em outras técnicas de tolerância a falhas, como migração de tarefas em diferentes recursos, já existente no serviço de provisionamento do BioNimbuZ, o qual, desta maneira, assegura que a tarefa interrompida possa ser reexecutada em outra máquina virtual em melhores condições. Outra sugestão é a implementação de uma monitoração proativa dos recursos, que permita prever indícios de falhas, que quando combinados com a técnica de *checkpoint*, seja possível migrar os processos antes que estes falhem. A análise do tempo ideal de *checkpoint* também é uma linha de pesquisa que pode ser aprofundada em trabalhos futuros.

Referências

[1] Ian, Foster: *Cloud computing and grid computing 360-degree compared.* Grid Computing Environments Workshop, páginas 1–10, 2008. ix, 7, 8

[2] C.N., Hofer: *Cloud computing services: taxonomy and comparison.* springerlink.com, página 83, 2011. ix, 9

[3] Rajkumar, Buya: *Inter-cloud architectures and application brokering: taxonomy and survey.* Wiley Online Library, páginas 378–382, 2012. ix, 10, 11

[4] Antonio, Celesti: *Three-phase cross-cloud federation model: The cloud sso authentication.* Second International Conference on Advances in Future Internet, páginas 94–101, 2010. ix, 11, 12

[5] Agarwal, Himanshu e Anju Sharma: *A comprehensive survey of fault tolerance techjmiques in cloud computing.* Conference on Computing and Network Communication, 2015. ix, 13, 14, 15, 16, 18

[6] Amin, Zeeshan, Nisha Sethi e Harshpreet Singh: *Review on fault tolerance techniques in cloud computing.* International Journal of Computer Applications, 2015. ix, 15, 16

[7] Liu, Yudan, Raja Nassar, Chokchai e Stephen L. Scott: *An optimal checkpoint/restart model for a large scale high performance computing system.* IEEE International Symposium on Parallel and Distributed Processing, 2008. ix, 17

[8] Egwutuoha, Ifeanyi P., David Levy, Bran Selic e Shiping Chen: *A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems.* Springer Science+Business Media New York, 2013. ix, 17, 18

[9] Almeida Ramos, Vinícius de: *Um sistema gerenciador de workflows científicos para a plataforma de nuvens federadas bionimbuz.* Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília, 2016. ix, 21, 22, 24, 26, 27, 30

[10] Moura, Breno Rodrigues de: *Arquitetura de um controlador de sla para ambiente de nuvens federadas.* Dissertação de mestrado, Departamento de ciência da computação, UnB Brasília, 2017. ix, 21, 22, 25

[11] Saldanha, Hugo, Edward Ribeiro, Maristela de Holanda, Aleteia Araújo, Genaina Rodrigues, Maria Emília Walter, Jõao Carlos Setubal e Alberto Dávila: *A cloud architecture for bioinformatics workflows.* International Conference on Cloud Computing and Services Science, 2011. 1, 21

[12] Tanembaum, Andrew S.: *Sistemam Operacionais Modernos.* Pearson, 2015. 4, 5, 6, 7

[13] Yeo, Che Shin, Rakjamar Buya, Houssein Pourresa, Rasit Eskicioglu, Peter Graham e Frank Sommers: *Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers.* Albert Y. Zo-maya, editor, Handbook of Nature-Inspired and Innovative Computing, páginas 521–551, 2006. 5

[14] Thomas, Sterling: *Beowulf: A parallel workstation for scientific computation.* In Proceedings of the 24th International Conference on Parallel Processing, páginas 11–14, 1995. 5

[15] Berman, G. F., Fox e A. J. G. Hey: *Grid computing: Making the global infrastructure a reality.* New York, NY, USA, 2003. 5

[16] Shirley, Radack: *Cloud computing: A review of features, benefits, and risks, and recommendations for secure, efficient implementations.* NIST ITL BULLETIN FOR JUNE 2012, páginas 2–3, 2012. 6

[17] Rimal, B. P., Eunmi, C., Lumb e I.: *A taxonomy and survey of cloud computing systems.* INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference, páginas 44–51, 2009. 7

[18] *Amazon ec2.* `https://aws.amazon.com/pt/ec2/`, acesso em 28.06.2018. 8

[19] *Google compute engine.* `https://cloud.google.com/compute/`, acesso em 28.06.2018. 8

[20] *Microsoft azure.* `https://azure.microsoft.com/pt-br/`, acesso em 28.06.2018. 8

[21] *Ibm bluemix.* `https://www.ibm.com/cloud-computing/bluemix/pt`, acesso em 28.06.2018. 8

[22] *Google app engine.* `https://cloud.google.com/appengine/`, acesso em 28.06.2018. 8

[23] *Google docs.* `https://www.google.com/docs/about/`, acesso em 28.06.2018. 8

[24] *Microsoft office 365.* `https://www.office.com/`, acesso em 28.06.2018. 8

[25] Krishna M, Kumar: *Multi data center cloud cluster federation – major challenges emerging solutions.* 2016 IEEE International Conference on Cloud Computing in Emerging Markets, página 1, 2016. 10

[26] Kumar, Krishna M, Dhilip S Kumar e Vinay Morudi: *Multi data center cloud cluster federation – major challenges emerging solutions.* IEEE International Conference on Cloud Computing in Emerging Markets, 2016. 10

[27] Antonio, Celesti: *How to enhance cloud architectures to enable cross-federation.* Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing, páginas 337–345, 2010. 10, 16

[28] Bilal, Kashif, Osman Khalid, Saif Ur Rehman Malik e Muhammad Usman Shahid Khan: *Fault tolerance in the cloud*. Encyclopedia on Cloud Computing, S. Murugesan and I. Bojanova Eds., IEEE Computer Press, Hoboken, NJ, USA, 2015. 13, 17

[29] Ataallah, Salma M. A., Salwa M. Nassar e Elsayed E. Hemayed: *Fault tolerance in cloud computing - survey*. IEEE International Symposium on Parallel and Distributed Processing, páginas 241–245, 2015. 13, 16

[30] Tanembaum, Andrew S. e Marteen Van Steen: *Distributed Systems Principles and Paradigms*. Pearson, 2001. 13

[31] Saikia, Lakshmi Prasad e Yumnam Langlen Devi: *Fault tolereane techniques and algorithms in cloud computing*. International Journal of Computer Science Communication Networks, 2014. 15

[32] Kaur, Jasbir e Supriya Kinger: *Analysis of different techniques used for fault tolerance*. International Journal of Computer Science and Information Technologies, 2014. 16

[33] Koren, Israel e C. Mani Krishna: *Fault Tolerance-Systems*. Elsevier, 2007. 17

[34] W., Van Der A Alst e Van Hee: *Workflow management: models, methods, and systems*. MIT press, 2004. 20

[35] Yu, J. e R Buya: *Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms*. Sci. Program. 14, 3,4, páginas 217–230, 2006. 20

[36] Oliveira Barreiros, Willian de: *Escalonador de tarefas para o plataforma de nuvens federadas bionimbuz usando beam search iterativo multiobjetivo*. Monografia, Departamento de Ciência da Computação, UnB, Brasília, 2016. 21, 28

[37] Rosa, Michel, Breno Moura, Guilherme Vergara, Edward Ribeiro e Lucas Santos: *Bionimbuz: A federated cloud platform for bioinformatics applications*. IEEE International Conference on Bioinformatics and Biomedicine, 2016. 21, 26

[38] Vergara, Guilherme Fay: *Arquitetura de um controlador de elasticidade para nuvens federadas*. Dissertação de mestrado, Departamento de ciência da computação, UnB Brasília, 2017. 21, 22

[39] Lima, Deric, Breno Moura, Gabriel Oliveira e Edward Ribeiro: *A storage policy for a hybrid federated cloud platform: a case study for bioinformatics*. IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2014. 22

[40] Azevedo, Diego Rodrigues e Tarcísio Batista de Freitas: *Biocirrus: Uma nova política de armazenamento para a plataforma bionimbuz de nuvem federada*. Monografia, Departamento de ciência da computação, UnB, Brasília, 2015. 22, 28

[41] Rosa, Michel: *Predição de tempo e dimensionamento de recursos para workflows científicos em nuvens federadas*. Dissertação de Mestrado, Departamento de Ciência da Computação, UnB, Brasília, 2017. 22

[42] Alves, Tiago Henrique Costa Rodrigues: *Uma arquitetura baseada em containers para workflows de bioinformática em nuvens federadas*. Dissertação de Mestrado, Departamento de Ciência da Computação, UnB, Brasília, 2017. 22

[43] Foundation, Apache: *Apache avro*. `https://avro.apache.org/`, acesso em 16.06.2018. 22

[44] Foundation, Apache: *Apache avro documentation*. `http://avro.apache.org/docs/1.8.1/`, acesso em 16.06.2018. 23

[45] Foundation, Apache: *Apache zookeeper*. `https://zookeeper.apache.org/`, acesso em 16.06.2018. 23, 28

[46] Junqueira, Flavio: *ZooKeeper: Distributed Process Coordination*. O'Reilly, 2014. 23

[47] Foundation, Apache: *Apache curator*. `https://curator.apache.org/`, acesso em 16.06.2018. 23

[48] Borges, Carlos Augusto Lima: *Escalonamento de tarefas em uma infraestrutura de computação em nuvem federada para aplicações em bioinformática*. Monografia, Departamento de ciência da computação, UnB, Brasília. 28

[49] Oliveira, Gabriel Silva Souza de: *Acosched - um escalonador para o ambiente de nuvem federada zoonimbus*. Monografia, Departamento de ciência da computação, UnB, Brasília, 2013. 28

[50] Moura, Breno Rodrigues e Deric Lima Barcelar: *Política para armazenamento de arquivos no zoonimbus*. Monografia, Departamento de ciência da computação, UnB, Brasília, 2013. 28

[51] Gallon, Ricardo Fernandes: *Política de armazenamento de dados em nuvens federadas para dados biológicos.* Tese de Mestrado, Departamento de ciência da computação, UnB, Brasília, 2014. 28

[52] Nuvem Federada BioNimbuZ, Novo Serviço de Armazenamento na Plataforma de: *Lucas facundo neiva santos*. Monografia, Departamento de ciência da computação, UnB, Brasília, 2016. 30

[53] Joshi, Nikhil A.: *A windowed adaptive trimming tool for fastq files using quality*. `https://github.com/najoshi/sickle`, acesso em 16.06.2018. 37

[54] D, Kim, Langmead B e Salzberg SL.: *Hisat2 graph-based alignment of next generation sequencing reads to a population of genomes*. `https://ccb.jhu.edu/software/hisat2/index.shtml`, acesso em 16.06.2018. 37

[55] Li, Heng e Bob Handsaker: *Sequence alignment/map - samtools*. `http://samtools.sourceforge.net/`, acesso em 16.06.2018. 37

[56] Anders, Simon: *Htseq: Analysing high-throughput sequencing data with python*. `https://htseq.readthedocs.io/en/release_0.10.0/`, acesso em 16.06.2018. 37

Apêndice A

Fichamento de Artigo Científico

Anexo I