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

Um Sistema Gerenciador de Workflows Científicos Para a Plataforma de Nuvens Federadas BioNimbuZ

Vinícius de Almeida Ramos

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

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

Brasília
2016

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

Um Sistema Gerenciador de Workflows Científicos Para a Plataforma de Nuvens Federadas BioNimbuZ

Vinícius de Almeida Ramos

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia da Computação

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

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

Prof. Dr. ª Genaína Nunes Rodrigues
CIC/UnB

Prof. Dr. Ricardo Zelenovsky
Coordenador do Curso de Engenharia da Computação

Brasília, 12 de fevereiro de 2016

Dedicatória

À minha família, amigos e linda esposa, Luiza Alencar, que sempre estiveram comigo me apoiando e dando forças.

Agradecimentos

Primeiramente, gostaria de agradecer à Deus pela iluminação e conhecimento que tem me concedido ao longo desses anos nessa caminhada. Também agradeço à minha família pelo amparo e amor, e pelas melhorias sugeridas neste trabalho. Agradeço em especial à minha amada esposa, Luiza Alencar, primeiramente por ter dito *SIM* no dia de nosso casamento e também por sempre me fazer sorrir, mesmo nos dias mais difíceis. À minha orientadora, Aletéia Patrícia, pela dedicação, conselhos, reuniões e motivação para continuar. Agradeço ao grupo do BioNimbuZ pela união e colaboração. Em especial ao Breno Moura, pela perseverança na solução de problemas e vontade de vencer os desafios que enfrentamos.

Por fim, agradeço imensamente aos precisos conselho do Edward Ribeiro, o qual sempre tinha a solução dos desafios que surgiam, pela disposição e enorme vontade de ajudar.

Resumo

A necessidade por maior poder de processamento e armazenamento, consequência da complexidade das atuais aplicações e sistemas, tem dado espaço para o desenvolvimento de novos paradigmas na Computação. Com isso, criou-se o conceito de Computação em Nuvem. Essa nova forma de prover serviços computacionais tem possibilitado o desenvolvimento e a criação de diversas aplicações que compartilham diferentes tecnologias e provedores de serviços. Neste cenário, aplicações em Bioinformática tem se beneficiado dessa nova plataforma, devido à exigência de quantidades cada vez maiores de processamento. O BioNimbuZ, plataforma de execução de *workflows* em Bioinformática desenvolvido na Universidade de Brasília, utiliza o paradigma de Computação em Nuvem para processar fluxos de aplicações em diferentes provedores de serviços computacionais, como Microsoft Azure e Amazon EC2. Dessa forma, faz-se necessário o gerenciamento da execução desses *workflows* desde sua submissão ao sistema até sua finalização, tal como o provimento de uma interface para que o usuário possa ter acesso a esses serviços. Este trabalho propõe melhorias no gerenciamento e controle dos *workflows* submetidos à plataforma BioNimbuZ, com o desenvolvimento de um Sistema Gerenciador de *Workflows* Científicos baseado em tecnologias *web*.

Palavras-chave: Computação em Nuvem, *Workflow* Científico, Sistema Gerenciador de *Workflows* Científicos, Tecnologias *web*

Abstract

The need for greater processing power and storage, caused by the complexity of today's applications and systems, has given space to the development of new paradigms in computing. Thus, it created the concept of Cloud Computing. This new way of providing computing services has enabled the development and creation of various applications that share different technologies and service providers. In this scenario, applications in Bioinformatics has benefited from this new platform due to demand increasing amounts of processing. The BioNimbuZ, a Bioinformatics workflow execution platform developed at the University of Brasilia, uses the Cloud Computing paradigm to process application flows in different computer services providers, such as Microsoft Azure and Amazon EC2. Thus, it is necessary to manage the execution of these flows (workflows) since its submission to the system until their completion, such as provide an interface for the user to have access to these services. This paper proposes improvements in the management and control of workflows undergoing BioNimbuZ platform, with the development of a Scientific Workflow Management System based on web technologies.

Keywords: Cloud Computing, Scientific Workflow, Scientific Workflow Management System, web technologies

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A busca por cada vez mais poder de processamento tem desenvolvido novos estudos e paradigmas na Computação. *Grids* computacionais, *clusters* e, mais atualmente, nuvens computacionais tem surgido como alternativa para suprir essa necessidade de maneiras distintas. Em 1969, Leonard Kleinrock [6], um dos cientistas que chefiou o projeto ARPANET (o qual se tornou a base da Internet) disse que a computação funcionaria a partir de um modelo como é visto atualmente na telefonia e na eletricidade, um serviço. Neste modelo, usuários acessam esses serviços independentemente de onde estão localizados ou de qual maneira são entregues, abstraindo diversas características do ambiente envolvido. Esse modelo se aproxima do conceito de Computação em Nuvem [6].

Entretanto, não existe na literatura consenso para a definição de computação em nuvem, porém algumas características fundamentais [66] estão presentes na maioria delas, tais como virtualização, escalabilidade, interoperabilidade, qualidade de serviço (QoS), gerenciamento de falhas, transparência e elasticidade. Esses aspectos fundamentais constituem o modelo de nuvens computacionais, em que o sistema passa a ilusão ao usuário de que este possui acesso a recursos ilimitados de software e hardware.

Nesse contexto, nasceu o conceito de Federação de Nuvens [7], o qual compreende o uso de diversos provedores em um único serviço. Isso provê características adicionais aos modelos anteriores de nuvem pública, privada e híbrida, tais como migração de recursos (como imagens de máquinas virtuais), redundância de dados, processamento paralelo, replicação de recursos, combinação de serviços complementares e fragmentação de dados. Adicionalmente, uma federação de nuvens possibilita o desenvolvimento de sistemas flexíveis e interoperáveis, o que diminui os custos de desenvolvimento e facilita sua expansão, com o custo de se adicionar complexidade ao sistema.

Nesse cenário, a Bioinformática tem se beneficiado com esse conceito de nuvens computacionais pela sua característica de tratar grandes quantidades de dados, produzidas pelas modernas máquinas que executam algoritmos de sequenciamento genômico. Dessa

forma, diversas ferramentas foram projetadas e implantadas tirando proveito dos recursos disponibilizados pela computação em nuvens. O BioNimbuZ, projeto desenvolvido na UnB [68], faz uso da infraestrutura de uma federação de nuvens para executar *workflows* de Bioinformática de maneira transparente, flexível, eficiente e tolerante à falhas, com acesso a grande poder de processamento e armazenamento.

Na Bioinformática, um *workflow* é um conjunto de diversas fases em que análises computacionais são executadas a partir de dados obtidos por meio de sequenciadores automáticos [68]. Cada pesquisa implica em uma combinação de diferentes ferramentas já existentes ou a serem desenvolvidas, o que adiciona complexidade ao sistema, pois torna-o mutável a cada nova pesquisa. Sistemas científicos que gerenciam *workflow* [49] devem automatizar a execução de *workflows* científicos, suportando usuários na criação, composição e verificação da execução do *workflow* gerado pelo usuário.

Nesse contexto, este trabalho propõe um sistema gerenciador de *workflows* científicos que trata o problema de manutenção dos *workflows* submetidos ao ambiente de nuvem federada BioNimbuZ, além de implementar uma interface baseada em tecnologias *web* para que os usuários possam acessar os serviços disponibilizados pelo BioNimbuZ através da Internet.

1.1 Motivação

Com o número crescente de sistemas computacionais utilizados na Bioinformática para execução de *workflows* científicos, fez-se necessária a criação de sistemas que gerenciem o ciclo de vida destes *workflows*. Este ciclo de vida é composto por cinco fases, que são: Criação e Composição, Planejamento de Recursos, Execução, Análise da Execução e Compartilhamento de Resultados [49].

Diante desse contexto, percebeu-se a necessidade de integração de um sistema gerenciador de *workflows* científicos à plataforma de nuvem federada BioNimbuZ.

1.2 Problema

Atualmente, para utilizar a plataforma BioNimbuZ e executar um fluxo de trabalho, ou *workflow*, o usuário deve seguir uma série de passos para atingir o resultado esperado de uma submissão de *jobs* à plataforma. Não existe o conceito de *workflows*, pois esses passos devem ser executados de maneira sequencial, desde a escolha do software computacional à indicação de suas entradas e saídas, repetindo o processo até que todo o fluxo de dados seja computado. Todo esse processo é realizado sem uma interface gráfica eficiente, pois

é necessário submeter comandos em um *terminal*, o que pode ser um empecilho para usuários sem conhecimento dessa ferramenta.

1.3 Objetivo

1.3.1 Principal

Propor e implementar um Sistema Gerenciador de *Workflows* Científicos acessível pela Internet, para que o usuário possa entrar de maneira segura no sistema, compor um *workflow* de maneira gráfica, enviar arquivos necessários à sua execução, integrar essa submissão ao núcleo do BioNimbuZ para ser executado, salvar seu estado para que o usuário tenha acesso posterior e, por fim, permitir que o usuário visualize o resultado para realizar pós-análise.

1.3.2 Específicos

Para propor e implementar um Sistema Gerenciador de *Workflows* Científicos e cumprir o objetivo principal definido na Seção 1.3.1, este trabalho tem os seguintes objetivos específicos:

- Desenvolver uma aplicação *web* utilizando a linguagem de programação Java para permitir que usuários possam acessar os recursos disponíveis pela plataforma BioNimbuZ;
- Desenvolver uma camada de integração utilizando *webservices REST* (*Representational State Transfer*) para transferir arquivos, recuperar dados, solicitar a execução de *workflows*, entre outras funcionalidades, para o núcleo do BioNimbuZ;
- Desenvolver mecanismos que possibilitem aos usuários do sistema desenhar, compor, iniciar a execução de *workflows* e verificar os resultados obtidos;
- Utilizar Bancos de Dados *MySQL* para prover a persistência dos dados enviados ao BioNimbuZ.

1.4 Organização do Trabalho

Este trabalho foi dividido em mais seis capítulos os quais são descritos a seguir: No Capítulo 2 são apresentados os conceitos básicos para compreensão do contexto atual da Computação em Nuvem, como a concepção de Sistemas Distribuídos, *Clusters* e *Grids*

computacionais. Além disso, expõe algumas tecnologias utilizadas em nuvens computacionais e detalha os principais aspectos da Federação de Nuvens.

No Capítulo 3 será abordado o conceito de *workflows* científicos, suas características e ciclo de vida. Também serão apresentados exemplos de Sistemas Gerenciadores de *Workflows* Científicos que nortearam o desenvolvimento do presente trabalho.

O Capítulo 4 detalha a plataforma de nuvens federadas BioNimbuZ, retratando sua arquitetura anterior ao desenvolvimento deste trabalho. Também serão descritas tecnologias utilizadas em sua construção e que otimizaram pontos como tolerância à falhas e comunicação entre componentes de sua arquitetura.

O Capítulo 5 apresenta o Sistema Gerenciador de *Workflows* Científicos proposto neste trabalho, com uma nova proposta de arquitetura para o BioNimbuZ. Nele serão detalhadas sua implementação, justificadas as escolhas das tecnologias utilizadas, bem como comparadas a antiga maneira de se acessar os serviços providos pelo BioNimbuZ com o novo método, baseado em tecnologias *web*.

No Capítulo 6 são mostrados os resultados obtidos com o desenvolvimento do Sistema Gerenciador de *Workflows* Científicos para o BioNimbuZ.

E por fim, no Capítulo 7 são apresentadas as conclusões obtidas e demais trabalhos futuros, que poderão melhorar e otimizar o proposto no presente trabalho.

Capítulo 2

Computação em Nuvem

O objetivo deste capítulo é mostrar os conceitos envolvidos no paradigma de computação distribuída, mais especificamente os de Computação em Nuvem, abordando suas características e suas particularidades. Também descreve as chamadas Federações de Nuvens Computacionais e como este trabalho pretende utilizá-las para integrar um Sistema Gerenciador de *Workflows* Científicos à atual plataforma do BioNimbuZ. Para isso, a Seção 2.1 mostra o histórico da computação distribuída e seus principais elementos (como *Grids* e *Clusters*). A Seção 2.2 apresenta os conceitos básicos que norteiam o entendimento sobre nuvens computacionais, mostrando como diversas pesquisas contribuíram para o conceito atual desta tecnologia. Por fim, a Seção 2.3 mostra os detalhes da chamada Federação de Nuvens Computacionais, quais seus objetivos, suas características e o que a distingue dos outros modelos de computação distribuídas.

2.1 Sistemas Distribuídos

A Internet é utilizada por bilhões de pessoas com vários propósitos diferentes, como ler e-mails, visualizar conteúdo multimídia, fazer compras ou apenas realizar uma busca por um assunto de interesse. Isso passa ao usuário a ilusão de que a informação e o sistema que a provê se encontram localmente em sua máquina. Mas, a Internet representa um enorme sistema distribuído que se parece como um recurso único, disponível em um conjunto de configuração de conexão e que está acessível em apenas poucos cliques [24].

O conceito de sistema distribuído possui diversas definições e pontos de vista. Colouris [10] define um sistema distribuído como "*um sistema em que componentes de hardware e software comunicam e coordenam suas ações apenas trocando mensagens*". Por outro lado, Tanenbaum [77] o define como "*uma coleção de computadores independentes que aparecem ao usuário do sistema como um único computador*". E Keith Marzullo [3], define um sistema distribuído como "*uma coleção de processos sequenciais* $P_1, P_2, ..., P_n$

e uma rede capaz de implementar canais de comunicação unidirecionais entre pares de processos para troca de mensagem".

Dessa forma, as definições convergem para um conjunto de aspectos comuns e essenciais que nos ajuda a distinguir sistemas distribuídos:

- São formados por um conjunto de computadores (ou unidades de processamento);
- São conectados por uma rede, sua comunicação é feita através de troca de mensagens e, portanto, não compartilham memória;
- São vistos pelos usuários como um recurso único;
- Facilitam a utilização de recursos por usuários (ou aplicações);
- São escaláveis, isto é, possibilitam o aumento ou a diminuição de usuários ou recursos de maneira facilitada;
- Possuem desafios de temporização e sincronização.

  Assim, embora seja possível perceber os desafios inerentes à implementação deste tipo de sistema, o ganho de poder computacional vale à pena esse esforço.

Nas próximas seções serão apresentados os conceitos e os principais aspectos de um *Cluster* computacional e de um *Grid*, os quais são tidos como base para o conceito de Nuvem Computacional.

2.1.1 *Cluster* Computacional

A primeira iniciativa na construção do conceito e implementação de *Clusters* Computacionais foi realizada pela IBM em meados dos anos 60 como alternativa de interligar grandes *mainframes* para prover aos seus usuários uma forma comercial mais eficiente de paralelismo. Contudo, o conceito de clusterização não ascendeu como previsto até o surgimento de outras três tecnologias: microprocessadores de alta-performance, redes de alta velocidade e protocolos para computação distribuída de alta-performance [85].

Os recentes avanços dessas três tecnologias, somadas à sua acessibilidade (baixo preço decorrente da alta demanda) facilitaram a implementação de *clusters* computacionais como solução do antigo problema da eficiência de custos na busca de sistemas paralelos de alta performance. Contudo, para se construir um *cluster*, os computadores componentes devem ter alguns elementos essenciais [54]:

- Vários computadores de alta performance (Servidores, *Workstations*, *Mainframes*);
- Sistemas Operacionais;

- Conexão com uma rede de alta performance (como *Gigabit Ethernet*);
- *Middleware* de gerenciamento de *clusters* (serviços e abstrações que facilitam o desenvolvimento de aplicações distribuídas). O *middleware* permite ao *cluster* manter a uniformidade na presença de diferentes hardwares e SOs;
- Ambiente de computação paralela;
- Aplicações.

A Figura 2.1 apresenta a arquitetura conceitual de um *cluster*, mostrando suas camadas e alguns dos elementos descritos acima:

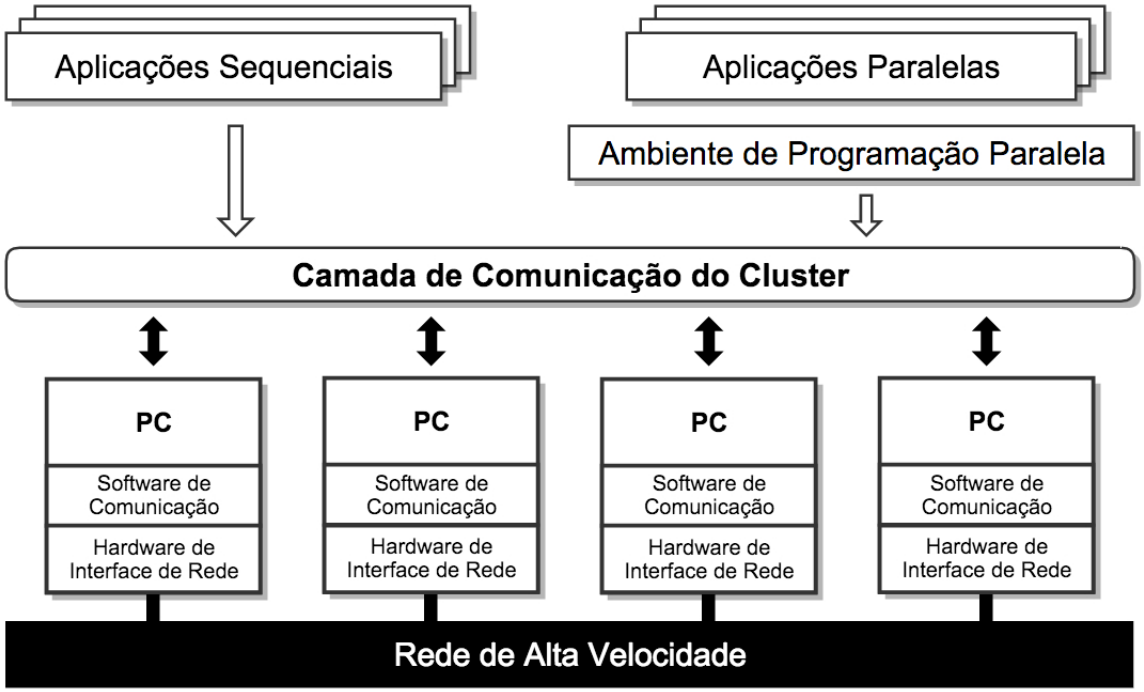

Figura 2.1: Arquitetura de um *Cluster*, adaptada de [54].

Na busca para provar o ganho de performance de *clusters* sobre as plataformas tradicionais de sistemas distribuídos, diversos projetos acadêmicos surgiram, como o Beowulf [54], o Berkeley NOW [87] e o HPVM [52].

Beowulf foi um *cluster* construído em 1994 por Thomas Sterling e Don Becker e consistiam de 16 processadores DX4 conectados por um canal Ethernet, dedicados à programação paralela [54]. O *cluster* criado foi muito bem visto pela área acadêmica e comercial, tanto que a NASA se interessou por este novo modelo. Em pouco tempo, o *cluster* Beowulf foi bastante difundido, tornando-se "Projeto Bewoulf" e passou a ser visto como gênero dentro da comunidade de Computação de Alta Performance (*High Performance Computing*).

Berkeley NOW (*Network of Workstations*) difere do *cluster* Beowulf pois seus nós computacionais eram computadores completos - computadores pessoais com teclado, mouse e som - conectados pela Internet (os nós de um *cluster* Beowulf são nós especializados, *workstations* que ficam dia e noite processando informações relacionadas ao *cluster*, não servindo, portanto, como um computador pessoal). Na maioria dos casos, nós NOW são utilizados para processamento a noite ou nos fins de semana, quando não estão sendo acessados pelos usuários, ou durante o dia, quando o *cluster* utiliza parte do poder de processamento do computador pessoal do usuário, utilizando ciclos ociosos de CPU e agregando-os pela Internet.

Esse modelo de computação distribuída tinha diversas limitações quanto ao tipo de aplicação que poderia ser executado no cluster NOW. Nesse contexto, o *cluster* Bewoulf tinha um melhor desempenho pois possuía as seguintes características:

- Processadores dedicados;
- Redes privadas de alta performance;
- Software customizável;
- Softwares clonados e enviados pela internet.

2.1.2 *Grid* Computacional

Em meados dos anos 90 o termo *Grid* Computacional foi criado para descrever tecnologias que possibilitariam usuários obterem poder computacional sobre demanda, assim como o serviço de energia é disponibilizado pela rede elétrica. O *grid* computacional distingue do modelo tradicional de computação distribuída pelo foco em compartilhamento de grandes quantidades de recursos e, geralmente, por ser orientado à computação de alta performance. É uma forma de computação que envolve coordenar e compartilhar recursos computacionais, aplicações, dados, armazenamento e/ou recursos de rede através de sistemas dinâmicos e geograficamente dispersos [35], de maneira flexível, segura e coordenada servindo às chamadas Organizações Virtuais [23].

Uma Organização Virtual é um conjunto de indivíduos e/ou instituições interessados em acessar diretamente máquinas, software, dados e outros recursos. Esse compartilhamento é extremamente controlado, com regras definindo o que está, com quem e as condições daquilo que está sendo compartilhado.

Um sistema gerenciador de um *grid* computacional não está somente voltado ao processamento de dados, mas também ao gerenciamento de recursos de todo o sistema, seja ele software (aplicações, protocolos, sistemas operacionais) ou hardware (armazenamento, consumo de CPU, utilização de memória). Foster *et al.* [24] dividem sua arquitetura em

cinco camadas, tomando como base a arquitetura do Protocolo de Internet (*Internet Protocol*). Essa arquitetura é apresentada na Figura 2.2, na qual estão presentes as seguintes camadas:

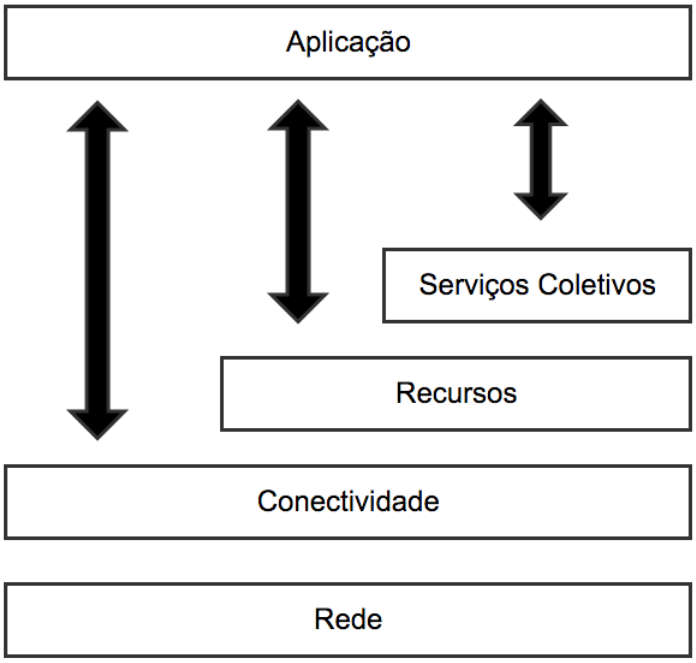

Figura 2.2: Arquitetura de um *Grid*, adaptado de [24].

- **Camada de Rede:** Fornece acesso compartilhado aos recursos computacionais a partir dos protocolos de *grid*. Um recurso pode ser uma entidade lógica, como um sistema de arquivos distribuídos, por exemplo um *cluster*;
- **Camada de Conectividade:** Define o método de comunicação e protocolos de autenticação para transações especifícas da rede interna do *grid*. Além disso, provê mecanismos de segurança criptográfica para verificar a identidade de usuários e recursos;
- **Camada de Recursos:** Define protocolos para transações, inicialização, monitoração, controle e custos de operações que utilizam recursos individuais. Esta camada se preocupa apenas com o seu conjunto de recursos, não se importando com o estado global dos recursos compartilhados;
- **Camada de Serviços Coletivos:** Enquanto a Camada de Recursos está ligada à interações com recursos únicos, a Camada de Serviços Coletivos tem o objetivo de capturar e gerenciar transações entre conjuntos de recursos;
- **Camada de Aplicação:** A última camada na arquitetura de *grids* abrange as aplicações que operam dentro do ambiente de uma Organização Virtual, ou seja,

esta camada tem como objetivo gerenciar as aplicações clientes que são executadas dentro de um grupo que compartilha o mesmo recurso.

Na próxima Seção serão apresentados os principais conceitos sobre Nuvens Computacionais, além das definições presentes na bibliografia, suas vantagens e desvantagens sobre os modelos de computação distribuída.

2.2 Nuvem Computacional

Com o rápido desenvolvimento de tecnologias de processamento e de armazenamento de dados e o crescimento da Internet, recursos de computação tornaram-se mais acessíveis, mais poderosos e mais disponíveis do que nunca. Essa tendência tecnológica permitiu a realização de um novo modelo de computação chamada computação em nuvem, em que os recursos computacionais (CPU, rede, memória, armazenamento, etc) são fornecidos como serviços que podem ser alugados e liberado pelos usuários através da Internet em um modelo sob-demanda. Esse conjunto de tecnologia provê aos usuários uma gama enorme de opções de usos dessa nova plataforma. Com ela, os serviços são disponibilizados de forma transparente, representando uma nova maneira de se utilizar recursos computacionais.

Porém, a forma de se prover poder computacional através de um modelo de computação baseado em serviços, assim como ocorre nas redes elétricas por exemplo, só foi possível quando grandes empresas capazes de implantar enormes *datacenters* a custos competitivos resolveram adotar esse novo paradigma da computação. Em 2006, o termo "nuvem" ganhou popularidade quando o CEO (*Chief Executive Officer*) da Google, Eric Schmidt, o utilizou para descrever o modelo de negócios para prestação de serviços pela Internet. Desse momento em diante, o termo "nuvem" realmente começou a ficar popular com ações de *marketing* e com o nascimento de diversas empresas especializadas.

A partir deste ponto, diversas definições surgiram em busca de um padrão para o conceito de computação em nuvem. Foster *et al.* [24] a definem como um paradigma de computação distribuída em larga escala movida pela economia da indústria, na qual um grande conjunto de recursos computacionais são providos sob-demanda a usuários externos pela Internet. Isso reforça a ideia do McCarthy de computação provida como serviços. Armburst *et al.* [26] trazem a definição de computação em nuvem como a união das aplicações disponibilizadas como serviços pela Internet e o hardware nos *datacenters* que as provém.

Por outro lado, o NIST (*National Institute of Standards and Technology*) a define como "*um modelo que possibilita o acesso ubíquo, conveniente, sob-demanda a um conjunto configurável de recursos computacionais (redes, servidores, armazenamento, aplicações,*

seviços) que podem ser rapidamente provisionados e liberados com um esforço mínimo de gerenciamento ou interação do provedor de serviço" [52].

Na indústria de Tecnologia da Informação (TI), o surgimento do modelo de computação em nuvem tem causado um enorme impacto nos últimos anos, onde grandes empresas como Google [42], Amazon [2] e Microsoft [53] se esforçam para fornecer plataformas de nuvem cada vez mais poderosas, disponíveis, de confiança e com melhor custo-benefício. Ao mesmo tempo, empresas procuram reformular seus modelos de negócios para se utilizarem dos benefício deste novo paradigma. Dessa forma, a computação em nuvem fornece vários recursos interessantes que a torna atraente para empresas, tais como [81]:

- **Não necessita de investimento inicial**: A computação em nuvem usa um modelo de precificação em que o usuário paga apenas pelo recurso computacional utilizado. Um prestador de serviço não precisa investir na infraestrutura para começar a se utilizar dos ganhos tecnológicos de uma nuvem computacional. Ele simplesmente aluga os recursos de acordo com suas próprias necessidades e paga por aquilo que utilizar;
- **Reduz o custo operacional:** Recursos em um ambiente de nuvem podem ser rapidamente alocados e desalocados sob-demanda. Assim, um prestador de serviços não precisa mensurar sua capacidade de acordo com a carga de pico (carga máxima). Isso proporciona uma enorme economia uma vez que os recursos podem ser liberados para economizar custos de serviço quando a demanda for baixa;
- **Altamente escalável:** Provedores de infraestrutura possuem grandes quantidades de recursos a partir de *datacenters* e os tornam facilmente acessíveis. Um provedor pode facilmente expandir sua capacidade, a fim de lidar com um rápido aumento na carga de serviço;
- **Acessibilidade:** Serviços hospedados na nuvem são geralmente baseado na *web*. Assim, são facilmente acessíveis através de uma grande variedade de dispositivos com conexões de Internet. Estes dispositivos não só incluem computadores *desktop* e *notebooks*, mas também celulares e *tablets*;
- **Reduz os riscos de negócios e despesas de manutenção**: Por terceirizarem os serviços de infraestrutura para as nuvens, prestadores de serviços transferem os riscos empresariais (tais como falhas de hardware) para os provedores de infraestrutura, que muitas vezes tem um maior conhecimento e estão melhor equipados para o gerenciamento desses riscos.

No entanto, embora a computação em nuvem tenha mostrado consideráveis oportunidades para a indústria de TI, também traz muitos desafios únicos que precisam ser

cuidadosamente abordados, tais como confidencialidade das informações, integridade de dados, disponibilidade dos serviços, tolerância a falhas e um dos pontos mais discutidos que é a segurança.

2.2.1 Tipos de Nuvens

As nuvens computacionais podem ser divididas em quatro tipos diferentes de implantação [52]: nuvens públicas, privadas, comunitárias ou híbridas, as quais são descritas a seguir:

- **Nuvens Públicas:** Sua infraestrutura é disponibilizada para o público em geral, mantida e gerenciada por uma instituição acadêmica, comercial ou governamental e não impõe condições para sua utilização. Seus recursos computacionais são provisionados dinamicamente e são acessíveis através da Internet (geralmente, com a utilização de *webservices*);
- **Nuvens Privadas:** A utilzação da infraestrutura desse tipo de nuvem é exclusivo de uma única companhia ou grupo de empresas, compreendendo múltiplos usuários. Seu gerenciamento pode ser feito pela própria instituição, por outra empresa ou pela junção de ambas as partes;
- **Nuvens Comunitárias:** Esse tipo de nuvem é provisionado para a utilização exclusiva de uma comunidade específica de consumidores de uma organização com interesses em comum (por exemplo, organizações com o mesmo requisito em segurança, mesmos requisitos em performance). Sua implantação pode ocorrer dentro ou fora da organização;
- **Nuvens Híbridas:** Esse tipo de infraestrutura é uma composição de dois ou mais tipos de nuvens (públicas, privadas ou comunitárias) em que o provedor de serviços disponibiliza tecnologias proprietárias que permitem portabilidade de dados e aplicações entre as nuvens que a compõe. Essa infraestrura é muito útil quando, por exemplo, se quer trafegar dados sensíveis e sigilosos em um ambiente controlado e gerenciado internamente (utilização de uma nuvem privada), enquanto outros dados e aplicações podem trafegar em nuvens públicas ou comunitárias.

2.2.2 Arquitetura

A arquitetura de uma nuvem computacional pode ser dividida em quatro camadas distintas [24][87]: camada de hardware (também chamada de camada de *datacenter*), camada de infraestrutura, camada de plataforma e a camada de aplicação. Elas podem ser descritas da seguinte maneira:

- **Camada de Hardware:** Esta camada é responsável pela gestão dos recursos físicos da nuvem, incluindo servidores, roteadores, *switches*, sistemas de energia e refrigeração. Na prática, a camada de hardware é implementada em enormes centros de dados (*datacenters*), os quais geralmente contém milhares de servidores organizados em *racks* e interconectado através de *switches*, roteadores ou outros dispositivos de rede. Problemas típicos na camada de hardware incluem configuração de hardware, tolerância a falhas, gestão, monitoração e refrigeração;
- **Camada de Infraestrutura:** Também conhecida como a camada de virtualização (*virtualization layer*). A camada de infraestrutura cria um *pool* de armazenamento e recursos de computação por meio do compartilhamento dos recursos físicos que utilizam tecnologias de virtualização como o Xen [26], o KVM [25] e o VMware [82]. A infraestrutura é uma camada essencial da computação em nuvem, uma vez que muitas características chaves, tais como a atribuição dinâmica de recursos, são apenas disponibilizadas através de tecnologias de virtualização;
- **Camada de Plataforma:** Construída no topo da camada de infraestrutura, a camada de plataforma consiste de sistemas operacionais e *frameworks* de aplicação. A finalidade da camada de plataforma é minimizar o ônus da implantação de aplicativos diretamente em *containers* VM (método em que a camada de virtualização é executada dentro do sistema operacional em que reside);
- **Camada de Aplicação:** Reside no nível mais alto da hierarquia, a camada de aplicação consiste nas aplicações reais sendo executadas em nuvem. Diferentes de aplicações tradicionais, aplicações em nuvem podem aproveitar o recurso de escalonamento automático para alcançar melhor desempenho, disponibilidade e custo operacional mais baixo.

2.2.3 Modelos de Serviço

A computação em nuvem emprega um modelo de negócios orientado a serviços, ou seja, recursos de hardware e de software são disponibilizados sob-demanda. Conceitualmente, cada camada da arquitetura, descrita na Seção 2.2.2, pode ser implementada como um serviço à camada acima. Em outras palavras, a camada acima será a consumidora dos serviços providos pela camada imediatamente abaixo. Esse modelo em camadas está descrito na Figura 2.3.

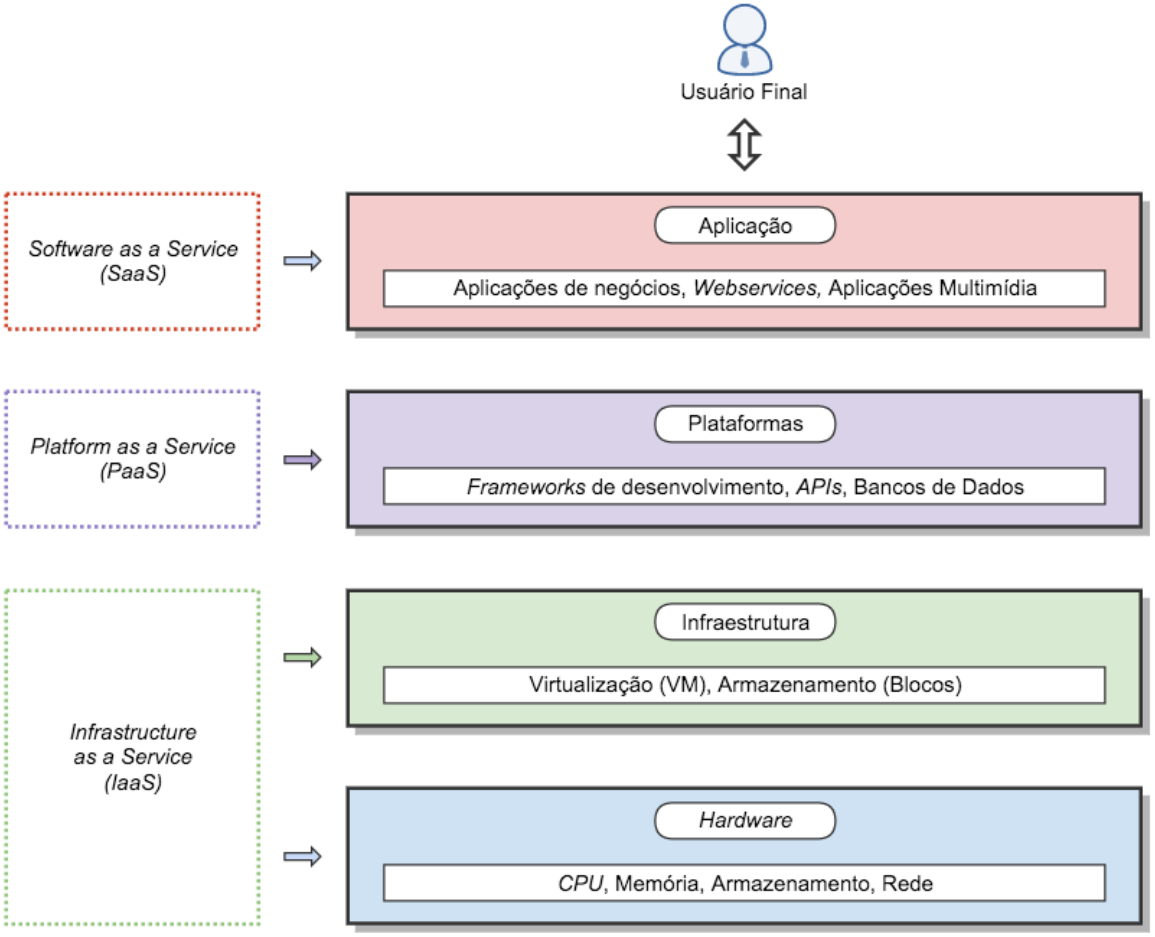

Figura 2.3: Arquitetura de uma Nuvem Computacional, adaptada de [87].

Esses serviços são categorizados no seguintes três modelos [24] [81] [87]:

- ***Infrastructure-as-a-Service*** (IaaS): Provê recursos computacionais aos usuários, tais como poder computacional, armazenamento de dados e redes virtuais para que possam implantar e executar qualquer tipo de software. A tecnologia de virtualização é essencial para este modelo, pois dá ao provedor deste serviço a habilidade de que vários usuários possam compartilhar recursos de uma mesma máquina física. Exemplos de provedores deste modelo são: Amazon EC2 [39], Google Compute Engine [43] e GoGrid [12].
- ***Platform-as-a-Service*** (PaaS): Dá ao consumidor deste serviço a capacidade de implantar aplicações criadas pelo usuário ou adquiridas de terceiros, criadas a partir de linguagens de programação, bibliotecas de software, *APIs* (*Application Programming Interface*) e ferramentas suportadas pelo provedor. Nesse modelo, o usuário não gerencia aspectos do hardware e do software necessários para execução

da infraestrutura da nuvem. Alguns exemplos são: Heroku [69], CloudForge [9] e AppScale [40].

- ***Software-as-a-Service*** (SaaS): Disponibiliza aplicações que estão sendo executadas em algum ponto da Internet por um provedor IaaS, eliminando a necessidade de instalar e executar a aplicação no computador do usuário. São independentes de plataforma e, geralmente, provêem uma interface para que o usuário possa acessar e utilizar esse serviço. Algumas empresas que implementam esse modelo de serviço são: Abiquo [46], SalesForce [70] e Zendesk [86].

A próxima seção traz os conceitos de um novo modelo de estrutura de nuvens computacionais, a federação de nuvens, a qual traz inovações e satisfaz necessidades trazidas pelos modelos de nuvem descrito nesta seção.

2.3 Federação de Nuvens

Com o passar dos anos, diversas necessidades surgiram no contexto da computação em nuvem e a sua utilização de forma isolada passou a não ser mais suficiente para algumas aplicações. Assim, medidas foram tomadas como forma de suprí-las, tais como a criação de *datacenters* espalhados ao redor do mundo para aumentar a tolerância a falha de suas aplicações, prevendo possíveis quedas de seus servidores e a redundância de dados, visando manter a taxa de disponibilidade dos serviços providos. Dessa forma, integrar diferentes serviços de nuvens para continuar atendendo às expectativas de qualidade de serviço (*QoS - Quality of Service*) passou a ser um requisito necessário para continuar fornecendo serviços de forma rápida, eficiente e escalável.

Assim, a federação de nuvens é uma área de pesquisa interessante na computação em nuvem e está muito focada na continuidade do *QoS*, empenhando-se no desenvolvimento de mecanismos para manter o serviço disponibilizado e sua qualidade.

Em um cenário de federação de nuvens, cada provedor de nuvem é capaz de ampliar de forma transparente a sua própria quantidade de recursos de virtualização (ou seja, o aumento do número de máquinas virtuais instanciadas), requerindo maior poder computacional e capacidade de armazenamento para outras nuvens. Por conseguinte, o operador de nuvem é capaz de satisfazer qualquer solicitação de alocação enviado por seus clientes.

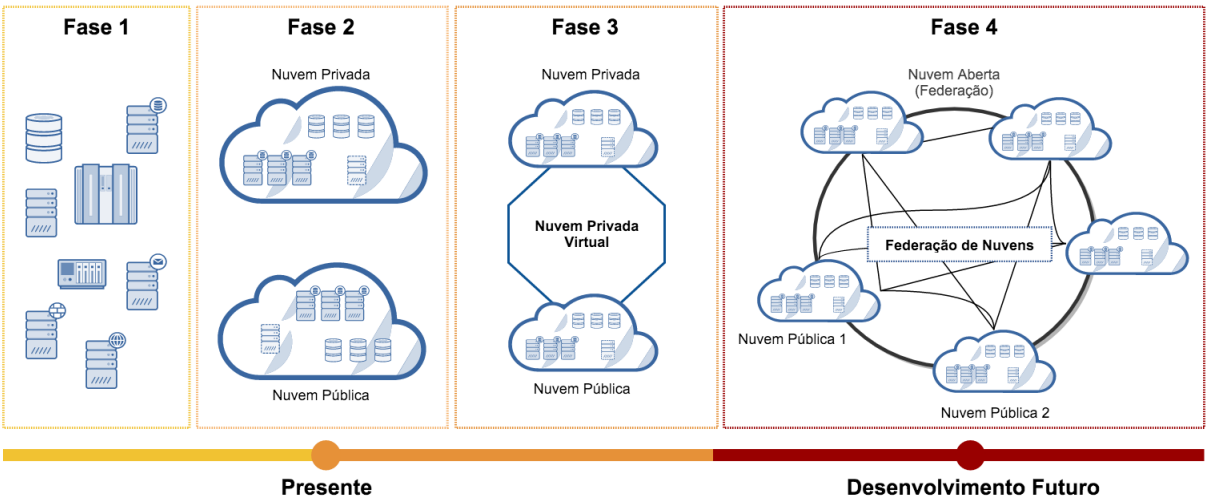

Figura 2.4: Fases da computação em nuvem, segundo White E. *et al.* [83].

Segundo [83], a federação de nuvens passou por quatro fases, as quais são claramente apresentadas na Figura 2.4, e descritas a seguir:

- **Primeira Fase:** É possível notar *datacenters* desconexos, sem utilizar o conceito de nuvem computacional, ou seja, estão espalhados pelo mundo e realizam todo o trabalho sozinhos, contando apenas com a sua própria infraestrutura e mecanismo de gerenciamento. Assim, estão mais propensos a falhas e não compartilham recursos computacionais com outros *datacenters*;
- **Segunda Fase:** Nessa fase, as nuvens já possuem tecnologia e mecanismos para disponibilizar seus serviços pela Internet, estando acessíveis a usuários finais. É o que é visto em serviços como Dropbox (*Software-as-a-Service*) [18], Amazon EC2 (*Infrastructure-as-a-Service*) [39] ou Heroku (*Platform-as-a-Service*) [69];
- **Terceira Fase:** As nuvens já conseguem compartilhar recursos entre si por meio de um provedor virtual que realiza todo o balanceamento das nuvens, de forma que nenhum provedor fique ocioso enquanto há outros que se encontram utilizando sua carga máxima de recursos. Celesti *et al.* [7] as tipificou em duas definições: *Home Clouds*, nuvens com sua capacidade computacional saturada e *Foreign Clouds*, aquelas com recursos ociosos disponíveis e que podem ser utilizados por outras nuvens. Nesse caso, uma solicitação seria enviada para a nuvem com recursos ociosos, e esses seriam disponilizados para a nuvem sobrecarregada;
- **Quarta Fase:** Atualmente os provedores de serviços estão nesta última fase, na qual encontram-se as várias federações de nuvens espalhadas pelo mundo, trabalhando

de forma interoperável entre si, ou seja, dividindo os seus recursos quando ociosos, e requisitando recursos quando saturados. Esse conceito de "nuvem de federações" forma um cenário no qual os recursos desperdiçados na nuvem são escassos, dada sua utilização por outros provedores. Isso traz um melhor balanceamento de carga, com poucos serviços indisponíveis por uma possível saturação, fazendo com que os usuários experimentem uma melhor experiência de aplicações hospedadas nas nuvens.

Apesar das vantagens desse novo modelo de computação em nuvem, a implementação de tal cenário de federação de nuvens não é trivial, pois as nuvens são mais complexas do que os sistemas tradicionais. Assim, uma federação de nuvens precisa atender aos seguintes requisitos [7]:

- **Automatismo e Escalabilidade:** Usando mecanismos de descoberta, a nuvem computacional que deseja fazer a integração com outra nuvem deve ser capaz de escolher a melhor nuvem que satisfaça as suas exigências de recursos;
- **Segurança Interoperável:** Se faz necessária a integração de diferentes tecnologias de segurança entre nuvens computacionais, permitindo uma nuvem ser capaz de juntar-se à federação sem alterar as suas próprias políticas de segurança.

2.4 Considerações Finais

Neste capítulo foram apresentados os principais conceitos relacionados ao histórico da computação distribuída, os quais basearam o atual entendimento sobre nuvens computacionais. Também foram apresentados os principais aspectos e característica de federação de nuvens.

O Capítulo 3 aborda conceitualmente *workflows* científicos e sistemas gerenciadores de *workflows* científicos, base para entendimento da solução proposta neste trabalho.

Capítulo 3

Workflow Científico

Neste capítulo serão apresentados os conceitos e principais aspectos relacionados à *Workflows*, à *Workflows* científicos, e como os chamados Sistemas Gerenciadores de *workflows* Científicos ajudam cientistas na construção destes fluxos de trabalho e no gerenciamento de seu ciclo de vida. Ainda apresenta alguns Sistemas de *Workflows*, mostrando suas características, assim como suas vantagens e desvantagens. Para isso, a Seção 3.1 mostra os conceitos básicos de um *workflow* científico. A Seção 3.2 apresenta o seu ciclo de vida, mostrando suas fases e principais características. A Seção 3.3 introduz os conceitos e definições de um Sistema Gerenciador de *Workflows* Científicos e apresenta alguns exemplos desses sistemas, consolidados nos meios acadêmicos e comercial. Por fim, na Seção 3.4 são descritas as vantagens percebidas ao se utilizar um Sistema Gerenciador de *Workflows* Científicos.

3.1 Conceitos e Definições

Um *workflow* é uma sequência de passos bem definidos usados para automatizar algum processo, de acordo com um conjunto de regras definidas. Nessa automação, documentos, informações ou tarefas são processados, de acordo com um conjunto de procedimentos. Assim, um *workflow* é composto de grupos de dados, fases de análise, fluxos e ferramentas ordenadas de maneira a se atingir o objetivo desejado.

Nesse contexto, existem diversos softwares computacionais, que utilizam uma gama enorme de tecnologias diferentes, que possuem a tarefa de gerenciar o ciclo de vida de um *workflow*. Esses sistemas são geralmente conhecidos como Sistemas Gerenciadores de *Workflow* (*WfMS - Workflow Management Systems*), e garantem que os passos que automatizam os processos ocorram na sequência correta.

No contexto da Bioinformática, as análises computacionais dos dados, obtidos por meio de sequenciadores automáticos 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 implica numa combinação diferente de ferramentas, de acordo com os objetivos da pesquisa, e este fluxo de passos é chamado de *workflow* de Bioinformática. Esse dinamismo gera uma complexidade adicional ao projeto, pois é necessário implementar mecanismos de gerenciamento dessa entidade (*workflow*) a cada nova pesquisa, assim como os dados utilizados em suas entradas (*inputs*) e saídas (*outputs*).

De acordo com Singh *et al.* [1] [74] um *workflow* científico é definido como "*uma série de atividades estruturadas e cálculos que surgem na resolução de problemas científicos*", e um "*processo automatizado que combina dados e processos em um conjunto estruturado de passos para implementar soluções computacionais para um problema científico*". Dessa forma, *workflows* científicos distinguem-se de *workflows* de negócio (*business workflows*) pois contém fluxos centrados nos dados [8] [48], são mais flexíveis e são utilizados principalmente para descrever a execução de experimentos científicos [71].

Um dos objetivos de se utilizar *workflows* científicos é apoiar e, sempre que possível, automatizar fases como visualização, tarefas repetitivas, acesso aos dados, transformação, análise e que, se feitas de outra maneira, estariam propensas a erros. Assim, *workflows* científicos são muitas vezes usados para encadear aplicativos especializados e de novos métodos de análise de dados.

No entanto, assim como ocorre em *workflows* de negócios, *workflows* científicos não são apenas mecanismos de gerenciamento da execução de uma sequência de passos. Outras áreas como modelagem, *design*, análise e reutilização de *workflows* estão se tornando cada vez mais importantes neste contexto. Assim, as principais vantagens ao se utilizar *workflows* científicos são: *(i)* auxiliar pesquisadores permitindo que eles se concentrem no domínio específico do seu trabalho, ao invés de perder tempo lidando com complexas questões de gestão de dados e de software complicados; e *(ii)* evitar desperdícios de poder computacional, otimizando a execução de *workflow* sobre os recursos disponíveis.

A seção seguinte trata da definição do ciclo de vida de um *workflow*, suas etapas e principais aspectos.

3.2 Ciclo de Vida de um *Workflow*

As várias fases e etapas associadas ao desenvolvimento, implantação e execução de *workflows* científicos compreendem o seu ciclo de vida [49]. Essas fases são em grande parte suportadas por sistemas gerenciadores de *workflows* existentes (*WfMS - Workflow Management System*), utilizando uma ampla variedade de abordagens e técnicas. A Figura 3.1 mostra o ciclo de vida de um *workflow*, e suas fases podem ser descritas da seguinte maneira:

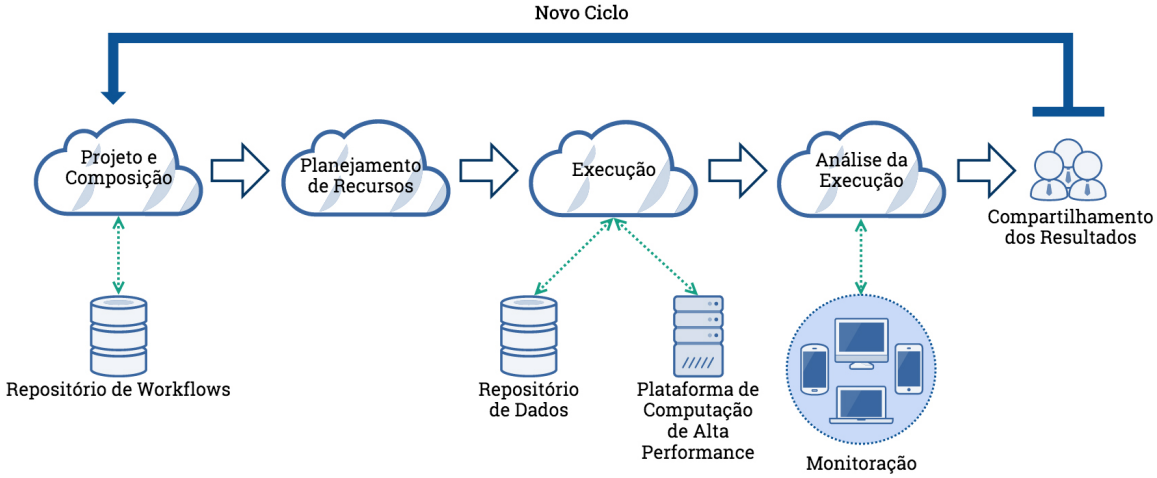

Figura 3.1: Ciclo de Vida de um *Workflow*.

- **Projeto e Composição:** O desenvolvimento de *workflows* científicos geralmente começa com o levantamento de requisitos dos cientístas envolvidos em uma pesquisa. Uma especificação das funcionalidades do *workflow* desejado é criada, e então um *workflow* é montado com base nesta especificação. O desenvolvimento de um *workflow* difere da programação tradicional de muitas maneiras. Normalmente, seu desenvolvimento abrange a composição e a configuração de um *workflow* para fins especiais de componentes, *subworkflow* e serviços pré-existentes. A construção de *workflows* assemelha-se mais à programação *script* do que do desenvolvimento de aplicações convencional. Durante a composição do *workflow*, o usuário ou cria um novo *workflow* modificando um existente, ou então compõe um novo *workflow* a partir do zero usando componentes e *subworkflows* obtidos a partir de um repositório externo. Com relação à representação interna de um *workflow*, sistemas de *workflow* científicos tendem a usar a sua própria linguagem e formatos de troca. As razões para esta divergência incluem a vasta gama de modelos de computação (computação em nuvem, por exemplo) utilizados na construção desses, e o foco do desenvolvimento das funcionalidades envolvidas, que no caso são orientadas para o cientista.

- **Planejamento de Recursos:** Assim que a descrição do *workflow* é construída, sistemas gerenciadores de *workflows* científicos muitas vezes fornecem várias funcionalidades antes da execução. Essas funções podem incluir a validação (por exemplo, verificação de tipo), a alocação de recursos, a programação, a otimização, os parâmetros de execução e a escolha de dados de entrada. Mapeamento de *workflows* é por vezes utilizado para se referir às decisões de otimização e programação feitas durante a fase de planejamento de recursos. Em particular, durante a fase de con-

cepção e composição, os recursos de destino a serem utilizados para a execução, não são tipicamente escolhidos. Mapeamento de fluxo de trabalho, então, refere-se ao processo de geração de um *workflow* executável baseada em uma descrição abstrata do *workflow* independente de recurso. Em alguns casos, o usuário executa o mapeamento diretamente escolhendo os recursos adequados. Em outros casos, o sistema gerenciador do *workflow* executa automaticamente o mapeamento. Neste último caso, os usuários tem permissão para construir *workflow* em um nível de abstração acima, não definindo aspectos do ambiente de execução.

- **Execução:** Uma vez que um *workflow* é mapeado, e dados foram selecionados e colocados à disposição do sistema gerenciador de *workflows*, o mesmo pode ser executado. Durante a execução, um sistema gerenciador de *workflow* pode registrar o histórico dos processos, bem como fornecer funções de monitoramento e de *failover* (recuperação em caso de falha) em tempo real. Dependendo do sistema, geralmente, é realizada a gravação dos passos que foram invocados durante a execução do *workflow*, os dados consumidos e produzidos por cada passo, um conjunto de dependências, e assim por diante. Um *workflow* pode mudar durante sua execução (por exemplo, devido a mudanças na disponibilidade de recursos), assim, a evolução deste *workflow* é dinâmica e deve ser registrada, a fim de dar suporte a análise posterior de sua execução.

- **Análise da Execução:** Após a execução do *workflow*, os cientistas envolvidos muitas vezes precisam inspecionar e interpretar os resultados gerados pelo *workflow*. Isso envolve a avaliação daquilo que foi produzido (ou seja, se o resultado faz sentido), examinar os rastros de execução do *workflow* (verificar o passo a passo da execução até se chegar ao resultado), depurar o *workflow* (verificar possíveis erros), e analisar o desempenho (examinar o tempo gasto por cada passo).

- **Compartilhamento dos Resultados:** Os dados e o produto do *workflow* podem ser publicados e compartilhados entre o grupo de pessoas envolvidas. A partir do momento em que os dados gerados pela execução do *workflow* são enviados para um repositório compartilhado, novas iterações do ciclo de vida do *workflow* podem começar novamente.

Outro aspecto muito importante na manutenção de *workflows* científicos é definir os papéis dos usuários do sistema (*User Roles*) [49]. Cada usuário possui uma função específica e bem definida no controle do ciclo de vida de um *workflow*. Dessa forma, os usuários de sistemas gerenciadores de *workflows* científicos podem desempenhar uma série de funções diferentes dentro das fases indicadas. Um *designer* de *workflows* é geralmente

um cientista que desenvolve um novo protocolo experimental ou analítico (ou uma nova variante de um método existente).

Como mencionado, um projeto de *workflow* é muitas vezes provocado por alguma forma de análise de requisitos. Assim, o desenho e os requisitos associados podem ser usados por um engenheiro de *workflows* para a implementação da descrição abstrata ou executável do *workflow* associado. Um operador do *workflow* é um usuário que executa *workflows* utilizando as entradas desejadas. Um operador pode iniciar um *workflow* diretamente através de um sistema gerenciador de *workflows* científicos, ou indiretamente através de outro aplicativo (por exemplo, dentro de um portal web); monitorar a sua execução (por exemplo, através de um *dashboard*) e, posteriormente, validar os resultados obtidos. Os papéis de usuário acima descritos não são necessariamente disjuntos. Por exemplo, uma única pessoa pode assumir os papéis de *designer*, de engenheiro e de operador do *workflow* [11].

Dessa forma, sistemas gerenciadores de *workflows* científicos visam tornar o projeto, a execução e o resultado da análise mais fácil em comparação com abordagens baseadas em *script* tradicionais para automação de processos científicos.

Na Seção 3.3 serão abordados características dos chamados sistemas gerenciadores de *workflows* científicio

3.3 Sistemas Gerenciadores de *Workflows* Científicos

A tecnologia da informação está revolucionando a forma que muitas pesquisas são conduzidas, com novas técnicas e modelos de computação, em campos multidisciplinares, tais como Bioinformática, Informática Biomédica, Quimioinformática, Geoinformática, etc. Para avançar ainda mais nessa nova ciência orientada a dados e à informação através da utilização de avançada infraestrutura de TI, grandes investimentos são realizados. Enquanto muitos esforços se concentram no desenvolvimento dessa infraestrutura, com novas tecnologias de *grid* e, mais recentemente, a utilização de nuvens computacionais, cientistas estão interessados em ferramentas, como bancos de dados distribuídos que permitam-lhes projetar, montar e executar seus próprios *workflows* científicos.

Idealmente, o cientista deve ser capaz de ligar qualquer recurso de dados científicos e serviços computacionais em um *workflow* científico, inspecionar e visualizar dados à medida que são processados, fazer alterações de parâmetros quando necessário, e executar novamente apenas os componentes afetados por um possível erro. Assim, um sistema de *workflow* científico torna-se um ambiente de resolução de problemas científicos, sintonizado cada vez mais com a utilização de infraestruturas de sistemas distribuídos como *grids* e nuvens computacionais [11].

Nesse cenário de utilização de infraestrutura de TI para otimizar o projeto, a execução e a análise de *workflows* científicos, diversas propostas de sistemas surgiram utilizando uma vasta gama de técnicas, infraestruturas e tecnologias, tais como: DiscoveryNet [34], Taverna [37], Triana [78], Kepler [48] e Pegasus [16] [73]. As próximas seções apresentam os principais sistemas gerenciadores de *workflow* apresentados na literatura.

3.3.1 *DiscoveryNet*

O sistema *Discovery Net* [11] foi concebido em torno de um modelo de *workflow* científico para a integração de fontes de dados distribuídos e ferramentas analíticas dentro de uma infraestrutura de *grid*. O sistema foi originalmente desenvolvido como parte do projeto financiado de *e-Science* do Reino Unido, *Discovery Net* (2001-2005) com o objetivo de produzir uma plataforma orientada para aplicações de alto nível, com foco na capacitação de cientistas na derivação de novos conhecimentos a partir de dispositivos, sensores, bancos de dados, componentes de análise e recursos computacionais acessíveis pela Internet.

O conjunto dedicado de componentes para a mineração de dados tem sido usado como base para inúmeros projetos de diversos domínios do conhecimento. Muitas das ideias de pesquisa desenvolvidas no âmbito do *Discovery Net* também foram incorporadas dentro do sistema IDBS (antigo InforSense KDE) [47], um sistema de mineração de dados e *workflows* de gerenciamento comercial que tem sido amplamente utilizada para aplicações de negócios orientados.

A Figura 3.2 fornece uma visão geral de alto nível do *Discovery Net*. O sistema é baseado em uma arquitetura multicamada, com um servidor de *workflows* fornecendo funções necessárias para a criação e a execução de *workflows*, tais como a integração e acesso a dados e recursos computacionais remotos, ferramentas de colaboração, publicação e mecanismos de visualização. O projeto do sistema tem como alvo os especialistas do domínio, ou seja, cientistas, ao invés do foco em desenvolvedores de sistemas distribuídos.

Os *workflows* criados desta forma também podem ser executados a partir de interfaces especializadas baseadas em tecnologias *web*. A implementação do sistema em si tem evoluído ao longo dos últimos anos, a partir de um protótipo direcionado a projetos específicos de um sistema de força industrial amplamente utilizado por organizações comerciais e acadêmicas.

3.3.2 *Taverna*

O principal objetivo do Taverna [37] é satisfazer as necessidades dos cientistas de Bioinformática que precisam construir *workflows* científicos a partir de inúmeros *webservices*

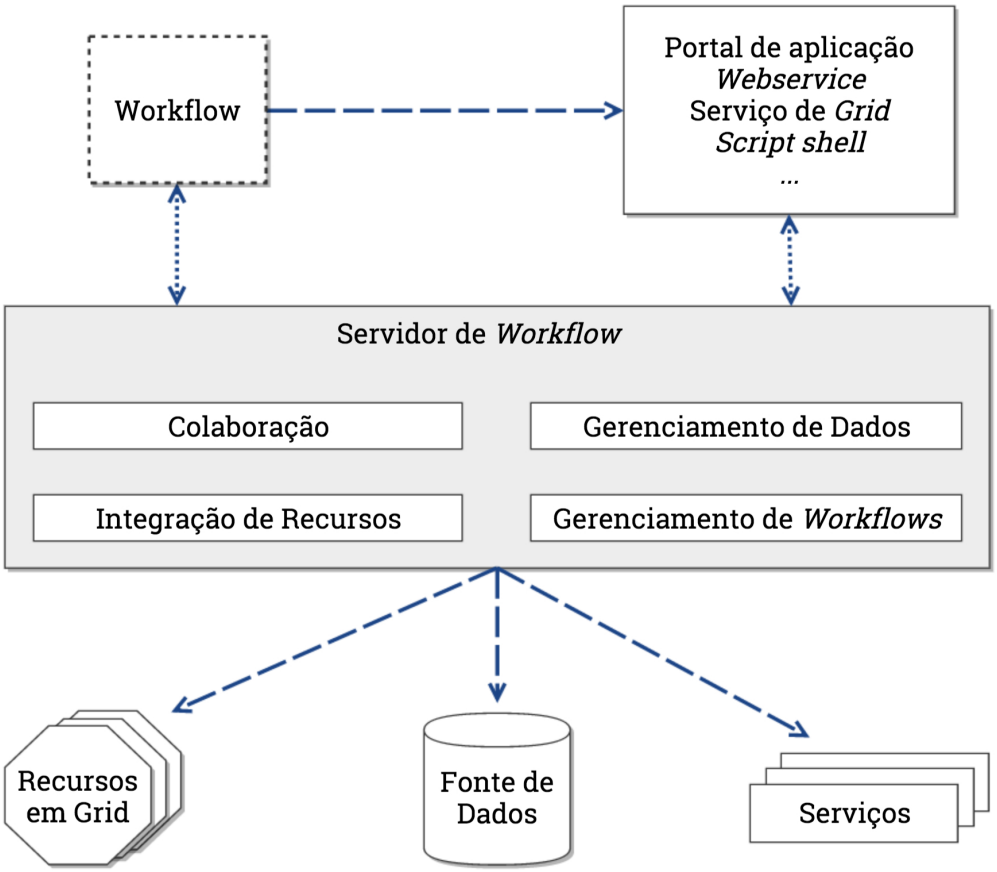

Figura 3.2: Arquitetura Multicamada do *DiscoveryNet*, adaptado de [11].

disponíveis remotamente através da Internet. Dessa forma, existe um esforço significativo no desenvolvimento do sistema Taverna para a aquisição e a organização desses *webservices* em uma coleção útil de componentes. Taverna foi construído utilizando-se o myGrid [13], que é um projeto de construção de *middleware* que suporta o desenvolvimento de *workflows* baseados em experimentos *in silico* (experimentos realizados através de simulações computacionais) de Biologia. Financiado pelo Programa *e-Science* do Reino Unido a partir de 2001, myGrid tem desenvolvido um conjunto de componentes de código aberto que pode ser usado de forma independente e em conjunto. Estes incluem:

- Um diretório de serviços, que são ferramentas de pesquisa atuantes em descrições semânticas de recursos e dados externos;
- Repositório de dados e metadados semanticamente dirigidos para a persistência dos dados de um *workflow* e seu ciclo de vida;
- Processamento distribuído de consultas (*queries*);
- Notificação de eventos.

Dessa forma, Taverna foi projetado para ser o ambiente de desenvolvimento e execução de *workflows* baseado em componentes do projeto myGrid.

Taverna é um nome comum usado para o sistema de *workflows* científicos que compreende o *Workbench* Taverna, que é uma ferramenta gráfica de criação de *workflows*; o servidor Taverna, utilizado para execução remota de *workflows*; seu conjunto de ferramentas de linha de comando (*Command Line Tool*) e, mais recentemente, o Taverna *Online*, que é a versão deste sistema disponível pela Internet. A Figura 3.3 mostra a interface percebida pelo usuário ao se utilizar o Taverna em sua versão *online*.

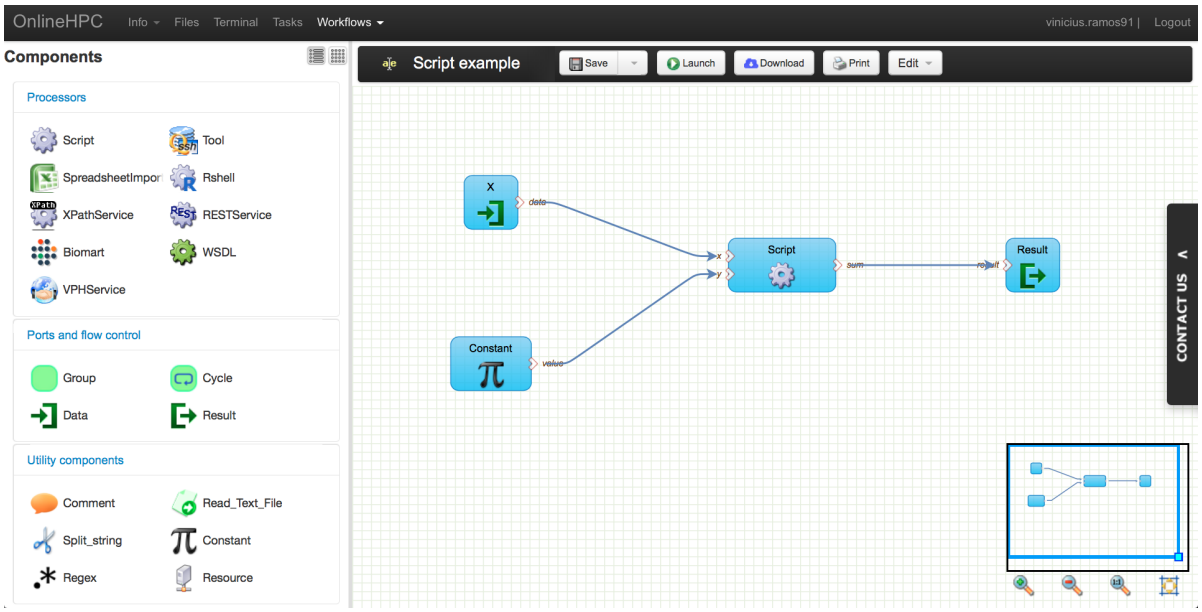

Figura 3.3: Ambiente Gráfico do Taverna *Online*, retirado de [57].

3.3.3 *Triana*

Triana [11] [50] é uma plataforma *open source*, distribuída, independente de plataforma, escrita na linguagem de programação Java utilizada como Ambiente de Resolução de Problemas (*PSE - Problem Solving Environment*). Um *PSE* é um ambiente de computação completo e integrado para composição, compilação e execução de aplicativos em uma área específica [33].

O Triana é um ambiente gráfico interativo que permite aos usuários compor aplicações e especificar seu comportamento de maneira distribuída. Conforme pode ser visto na Figura 3.4, o usuário cria um *workflow* arrastando as unidades desejadas para a área de trabalho e interliga estes arrastando uma ligação entre eles. Embora o foco seja a interface

gráfica, Triana é composto por um conjunto complexo de componentes de interação com potencial de criar sistemas completos ou qualquer subconjunto dele.

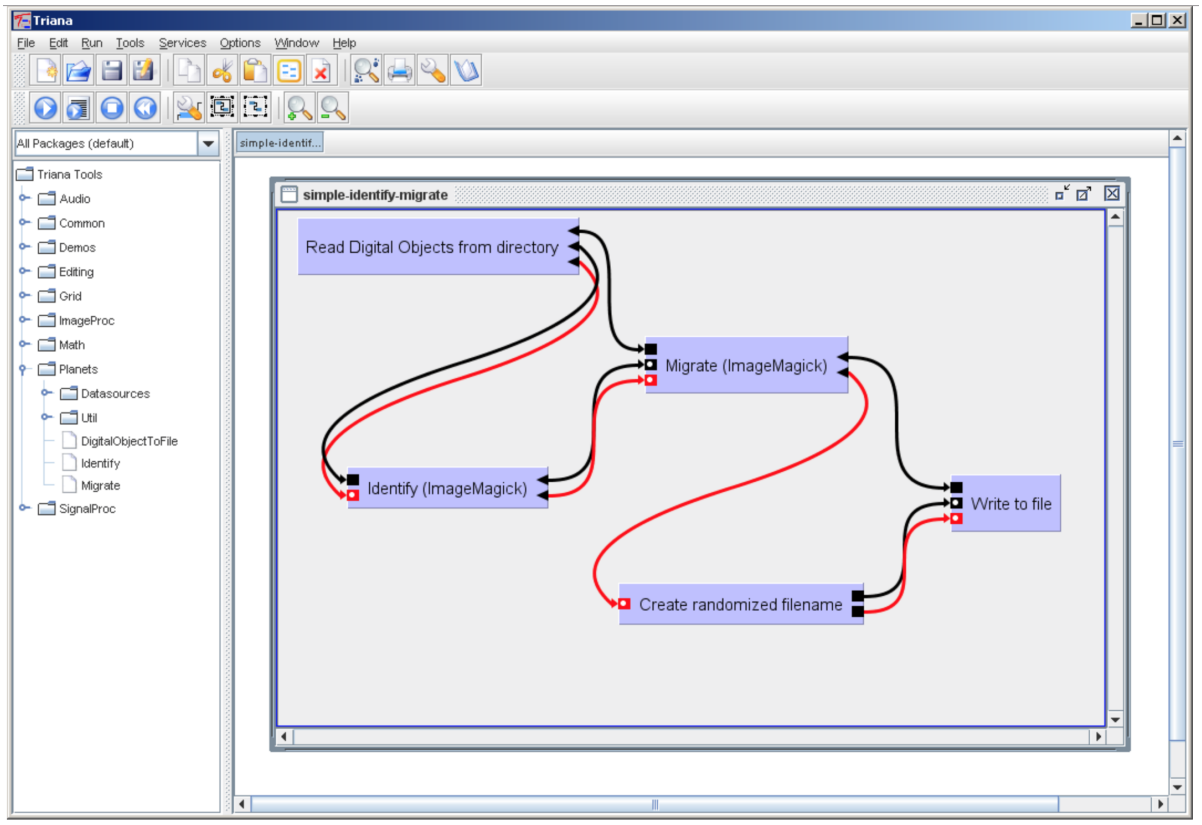

Figura 3.4: Ambiente Gráfico do Triana, retirado de [19].

Essa abordagem dá ao Projeto Triana a flexibilidade necessária para ser aplicado a muitos cenários diferentes e em muitos níveis diferentes. Por exemplo, ele pode ser usado como uma base para aplicações de *workflows* em *grids*, para conectar componentes de *grid* orientados a dados e gerenciar o *workflow* entre eles, ou como um sistema de análise de dados para aplicações de imagem, processamento de sinais ou texto, permitindo que um cientista possa aplicar rapidamente algoritmos em conjuntos de dados e visualizar os resultados.

Triana também pode ser usado como um editor gráfico de *scripts* de alto nível para a criação de *workflows* baseados em *scripts*.

3.3.4 *Kepler*

Kepler [48] é um sistema para construção, composição e mecanismo de orquestração de *workflows* científicos, desenvolvido a partir do Sistema *Ptolemy* II [5], que é uma ferramenta de modelagem orientada ao "ator" (*actor-oriented modeling tool*) [11].

O foco de Kepler é na análise de dados e na modelagem de maneira genérica. Assim, Kepler é adequado para modelar processos em uma ampla variedade de domínios científicos, desde modelos físicos até *webservices* de Bioinformática. Ao invés de tentar fornecer uma semântica genérica para todos os possíveis tipos de processos encontrados nestes domínios, Kepler separa o mecanismo de execução do *workflow* e atribui um modelo de computação, chamado diretor (*director*) [48], para cada *workflow*. Os componentes do *workflow* no Kepler são chamados de atores (*actors*) e representam operações ou fontes de dados, com um determinado número de portas, podendo ser entrada, saída, ou ambos, os quais atuam como pontos finais (*end-points*) para conexões. A interação entre esses componentes pode ser vista na Figura 3.5.

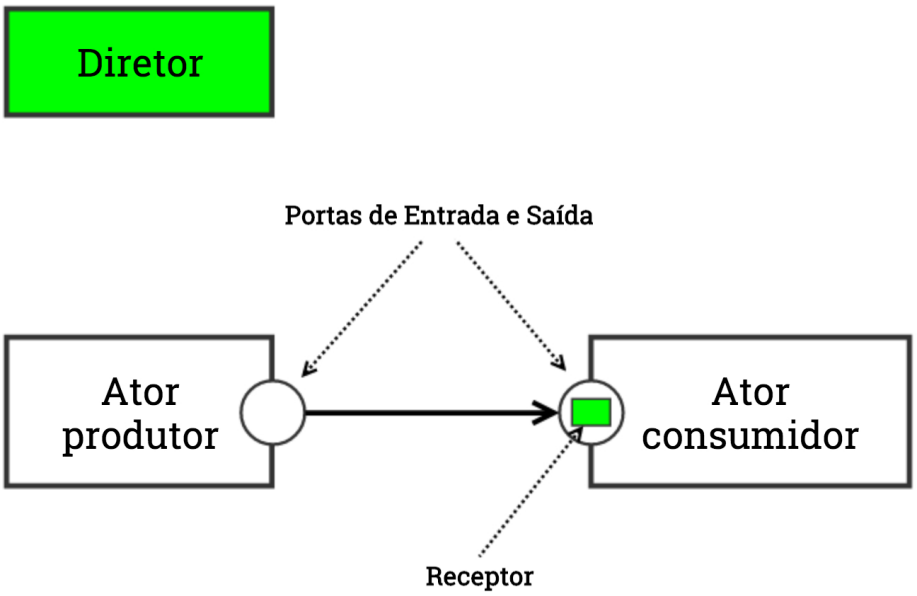

Figura 3.5: A Semântica da Interação entre os Componentes, adaptado de [48].

A interação mais simples consiste em um *actor* consumir um conjunto de dados sobre cada porta de entrada e produzir um sinal de saída sempre que ele executa. No entanto, pode ocorrer de um ator consumir ou produzir mais de um sinal para cada execução.

Director é um conceito-chave no sistema Kepler. Enquanto *actors* e relações, juntos, constituem um modelo de *workflow*, os *directors* se encontram em um nível acima, formando seu modelo de execução ou modelo de computação (*MoC - Model of Computation*) [48] . Nesta configuração, um *actor* só tem conhecimento da operação a ser executada e que saídas serão produzidas. A decisão de como e quando escalonar a execução de cada *actor* é deixada para o *director*. Por exemplo, uma operação de adição pode aceitar dados de entrada fornecidos por diversos mecanismos, incluindo eventos discretos, envio de dados e passagem assíncrona de mensagens (*asynchronous message passing*). Portanto, dependendo do *director* utilizado, os *actors* podem ter *threads* de controle separadas, ou

eles podem ter suas execuções desencadeada pela disponibilidade de uma nova entrada [48].

Dessa forma, usando Kepler, os cientistas podem capturar seus *workflows* em um formato que pode ser facilmente compartilhado, arquivado, versionado, e executado. A interface gráfica intuitiva de Kepler (herdada do sistema *Ptolemy*), a qual pode ser vista na Figura 3.6, e sua modelagem orientado à atores (*actor-oriented modeling*) o tornaram uma ferramenta muito versátil para o gerenciamento do ciclo de vida de um *workflow* científico para ambos engenheiros de *workflows* e usuários finais [48].

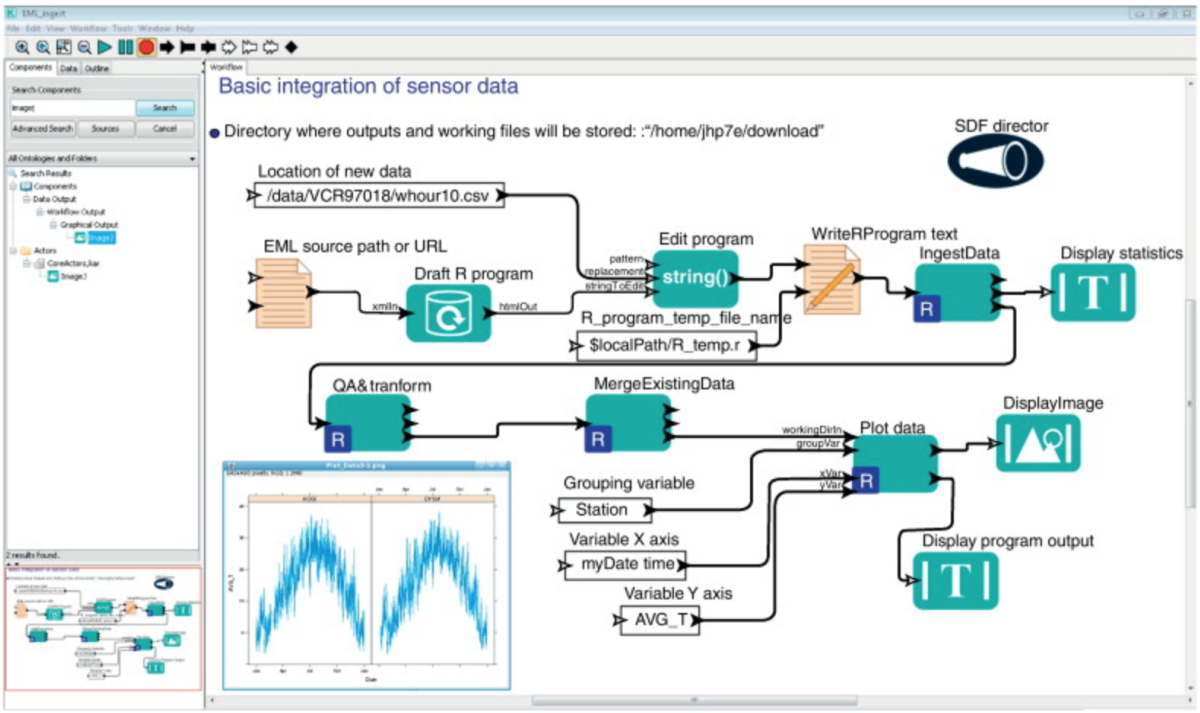

Figura 3.6: Interface Gráfica do Kepler, retirado de [41].

Os *workflows* gerados pelo sistema Kepler podem ser compartilhados a partir de uma representação XML (*Extended Markup Language*), gerado pela própria linguagem de modelagem do *Ptolemy* (*MOML - Modeling Markup Language*) [11].

3.3.5 *Galaxy*

O software ***Galaxy*** [22] é executado em servidores baseados em Linux/Unix, e fornece uma interface de usuário baseada em navegadores (ver Figura 3.7). O usuário final pode acessar Galaxy a partir de qualquer sistema operacional com um navegador *web*.

Ferramentas individuais oferecidas através do *Galaxy* são instalados e executados no servidor *Galaxy* ou em um *cluster* de computação associado, o que significa que o usuário final não tem que baixar ou instalar nenhuma ferramenta. Isso evita muitos problemas

práticos com a implantação e atualização de ferramentas computacionais, muitos das quais são complicadas de instalar e, em alguns casos, incompatíveis com o sistema operacional do usuário.

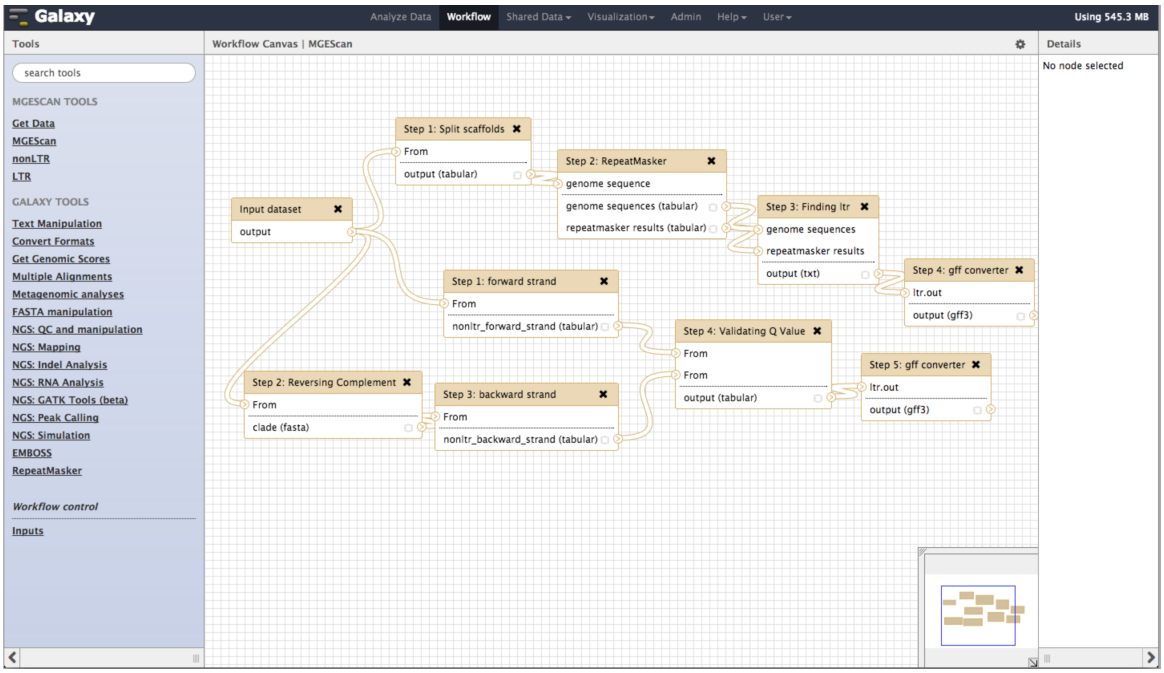

Figura 3.7: Interface Gráfica do Galaxy, retirado de [17].

Há um benefício de usabilidade adicional para esta abordagem baseada em navegadores, pois todas as ferramentas fornecidas através Galaxy são dipostas em uma interface gráfica consistente e familiar ao usuário. Isso contrasta com a alternativa de trabalhar com ferramentas de linha de comando através do teclado, o que pode gerar desconforto ao usuário que não possui prática em ferramentas baseadas em *terminal*.

Os desenvolvedores do *Galaxy* hospedam um servidor públic *Galaxy* através do site `http://usegalaxy.org`, o qual está localizado na Universidade Estadual da Pensilvânia (*PSU - Pennsylvania State University*). Esse oferece uma ampla gama de ferramentas instaladas. *Galaxy* também pode ser baixado e executado em um servidor local, que é a meneira seguida por muitos institutos, departamentos e grupos de pesquisa. Essa abordagem local é muito útil para pesquisadores que trabalham com ética ou com dados comercialmente sensíveis, para os quais o *upload* para a *PSU* (ou qualquer outro servidor) pode não ser possível.

O carregamento de grandes conjuntos de dados para um servidor também pode ser limitada por quotas de largura de banda e de usuário disponíveis, e uma instalação local também pode ser útil neste caso. Instâncias de servidores *Galaxy* locais podem ser configurados para executarem trabalhos em um *cluster* de computação local.

Finalmente, *Galaxy* também pode ser executado em uma plataforma de computação em nuvem, a partir da utilização de máquinas alugadas de um provedor de nuvem computacional, como a *Amazon EC2* [39], que oferece uma maneira fácil de ampliar o poder computacional alugado com base na demanda de utilização do software *Galaxy*.

3.4 Vantagens na Utilização de um Sistema Gerenciador de *Workflows* Científicos

Como dito anteriormente, *workflows* científicos são projetados para ajudar os cientistas a realizarem experimentos computacionais eficientes, proporcionando um ambiente que simplifica o desenho experimental, a implementação e a documentação. A crescente utilização de ambientes e sistemas de *workflows* científicos é devido a uma série de vantagens que estes sistemas apresentam, tais como:

- *Workflows* científicos automatizam tarefas repetitivas, permitindo que o usuário se concentre na ciência por trás do experimento, em vez de gastar tempo projetando o fluxo de dados e a gestão dos processos que compõem o *workflow*. Por exemplo, a repetição de um mesmo experimento com o intuito de se calibrar parâmetros, processo este que pode ser repetido milhares de vezes, pode ser alcançado muito mais facilmente através da utilização de sistemas de *workflows* do que com as abordagens convencionais de programação.

- *Workflows* científicos documentam, explicitamente, o processo científico que está sendo executado, o que pode levar a uma melhor comunicação entre, por exemplo, uma equipe de cientistas ou uma comunidade acadêmica, colaboração (por exemplo, o compartilhamento de *workflows* entre cientistas) e a reprodutibilidade dos resultados.

- Sistemas de *workflows* científicos podem ser usado para monitorar a execução de *workflows* e registrar a proveniência dos resultados de fluxo de trabalho. Proveniência, em particular, fornece uma forma de documentação que pode ser usado para validar e interpretar os resultados produzidos por (muitas vezes complexas) processos científicos.

- Sistemas de *workflows* científico muitas vezes podem otimizar e executar de forma mais eficiente os processos científicos, por exemplo, ao expor e explorar várias formas de paralelismo inerente nos processos científicos orientados por dados, bem como pelo emprego de outras técnicas de gestão eficiente dos recursos.

- Ambientes de execução de *workflows* incentivam a reutilização de artefatos de conhecimento (atores, fluxos de trabalho, conjuntos de dados, etc.) desenvolvidos ao se automatizar um processo científico, tanto dentro como entre disciplinas.

3.5 Considerações Finais

Neste capítulo foram retratados os principais conceitos para o entendimento de *workflows* científicos. Também foram apresentados diversos sistemas gerenciadores de *workflows* e suas funcionalidades, as quais serviram como referência da solução proposta neste trabalho.

O Capítulo 4 introduz as características da plataforma de federação de nuvens BioNimbuZ, apresentando sua arquitetura, anterior ao desenvolvimento deste trabalho, seus serviços e seus objetivos.

Capítulo 4

BioNimbuZ

Este Capítulo tem como objetivo apresentar as características da plataforma de federação de nuvens BioNimbuZ, no qual será desenvolvido o sistema gerenciador de *workflows* científicos projetado neste trabalho. Além disso, serão detalhados aspectos de sua arquitetura atual e o funcionamento dessa plataforma. Para isso, na Seção 4.1 serão apresentados os principais aspectos da plataforma BioNimbuZ, seus objetivos e uma visão geral de seu funcionamento. A Seção 4.2 traz a arquitetura do BioNimbuZ anterior ao desenvolvimento deste trabalho apresentando suas camadas e serviços componentes (uma nova arquitetura de implementação será apresentada no Capítulo 5).

4.1 Principais Características

O BioNimbuZ é uma plataforma de execução de *workflows* que utiliza uma infraestrutura provida por uma federação de nuvens híbridas proposta originalmente por Saldanha [68], e que tem sido aprimorada constantemente em outros trabalhos [13] [15]. O BioNimbuZ foi desenvolvido para suprir a demanda de plataformas de nuvens federadas, tendo em vista que a utilização de nuvens de forma isolada não atende, em muito casos, às necessidades de processamento e de armazenamento na execução das aplicações de Bioinformática.

A plataforma BioNimbuZ permite a federação de nuvens de diversos tipos, tanto privadas quanto públicas. Dessa forma, cada provedor pode manter suas características e políticas internas, e oferecer ao usuário transparência e ilusão de recursos infinitos. Assim, o usuário pode usufruir de diversos serviços sem se ter que gerenciar a infraestrutura que está sendo utilizada.

Um aspecto importante na arquitetura do BioNimbuZ é a flexibilidade na inclusão de novos provedores, pois são utilizados mecanismos desenvolvidos em sua implementação inicial, chamados *plugins* de integração. Estes tem como objetivo serem a interface de comunicação entre um provedor e os demais componentes da arquitetura BioNimbuZ, e

também entre ele e os demais provedores da federação. Para que a comunicação seja feita com sucesso, o *plugin* precisa mapear as requisicões vindas dos componentes do BioNimbuZ para acões correspondentes a serem realizadas na infraestrutura do provedor de serviço. Isso é fundamental para que requisitos como escalabilidade e flexibilidade possam ser alcançados.

Em sua implementação inicial, a comunicação realizada no BioNimbuZ era feita por meio de uma rede *Peer-to-Peer* (P2P) [76]. Porém, para alcançar os requisitos de escalabilidade e flexibilidade, percebeu-se a necessidade de alterar a forma de comunicação entre os componentes do BioNimbuZ, pois a utilização de uma rede de comunicação *Peer-to-Peer* (P2P) não estava mais suprindo as necessidades desses dois requisitos. É importante ressaltar que os outros objetivos propostos por Saldanha [68], tais como obter uma plataforma com grande poder computacional e armazenamento disponíveis, que suportasse uma diversidade de provedores homogêneos de infraestrutura e que fosse tolerante a falhas, foram mantidos e otimizados.

A fim de suprir essas necessidades, e também realizar a troca de mensagens de forma transparente, os trabalhos [13] [15] propuseram uma nova forma de comunicação, utilizando Chamadas de Procedimento Remoto (*RPC*) [79]. Isso possibilitou a chamada de procedimentos localizados remotamente sem que o usuário percebesse.

Nesse contexto, foi escolhido o *framework RPC* da Fundação Apache [16], e Avro [27], visando resolver também outros problemas de organização e de coordenação dos serviços disponibilizados na plataforma BioNimbuZ. Além disso, também foi utilizado um serviço voltado a sistemas distribuídos, chamado ZooKeeper [29], também da Fundação Apache. Assim, nas Seções 4.1.1 e 4.1.2 serão mostrados detalhes do funcionamento dos sistemas Apache Avro [16] e ZooKeeper [29].

4.1.1 Apache Avro

Um *framework RPC* é uma ferramenta utilizada para que máquinas possam realizar procedimentos em outras máquinas, de maneira remota. Esse conceito faz parte do paradigma de sistemas distribuídos e, possibilita ações como, por exemplo, a mesma instrução ser executada em conjuntos de dados diferentes em locais diferentes, ou uma máquina disparar a mesma requisição de procedimento em diversas máquinas geograficamente espalhadas.

O fato de ser um software livre, utilizando mais de um protocolo de transporte de dados em rede, e com suporte a mais de um formato de serialização de dados são algumas das vantagens na utilização do *framework* Avro. Quanto ao formato dos dados, ele dá suporte a dados binários e a dados no formato JSON [32]. Avro se baseia no protocolo HTTP [21] para realizar chamadas de procedimentos remotos.

O Projeto Avro foi criado com o objetivo de ser utilizado com um grande volume de dados. Esses dados dentro do *framework* Avro são representados como uma rica estrutura de dados com tipos primitivos (como inteiros, *strings*, caracteres, etc) e tipos complexos (como *unions*, *record*, *enum*, matrizes, mapas, etc), um formato de dados compacto, rápido e binário, e a integração de forma simples com diversas linguagens de programação. Essas características fazem do Avro uma ferramenta muito eficiente e leve, não onerando assim o poder computacional de sistemas distribuídos.

Em comparação com outros sistemas de chamadas *RPC*, como Thrift [28] e Protocol Buffers [44], o Avro difere nos seguintes aspectos fundamentais [16]:

- **Tipagem Dinâmica:** O Avro não requer que a sua representação dos dados seja gerada. Os dados são sempre acompanhados por um esboço (*schema*), que permite o processamento completo desses dados, sem a geração de código adicional, tipos de dados estáticos, etc. Isso facilita a construção de sistemas de processamento de dados e linguagens de genéricos.
- **Dados sem *Overhead*:** Como o *schema* do Avro está presente no momento de leitura dos dados, informações consideravelmente menores precisam ser codificadas juntamente com os dados. O Avro fornece um esquema com dados binários que permite que cada dado seja escrito sem sobrecarga (*overhead*), resultando em uma codificação de dados mais compacta, e, consequentemente, um processamento de dados mais rápido.
- **Atribuição Automática de ID's de Campos:** Quando um *schema* muda, o antigo e o novo estão sempre presentes durante o processamento dos dados, de modo que as diferenças possam ser resolvidas simbolicamente, usando-se os nomes de campo atribuídos pelo próprio Avro, o que resulta em uma maior flexibilidade no desenvolvimento de soluções, tendo em vista que o usuário não fica preso à um *schema* antigo, ou legado.

Portanto, o *framework RPC* Avro foi escolhido como *middleware* de Chamadas Remotas de Procedimentos para o BioNimbuZ, por ser livre, flexível, eficiente e possuir integração com várias linguagens.

4.1.2 Apache ZooKeeper

Apache ZooKeeper [29] é um serviço distribuído de coordenação de serviços, de código aberto, geralmente utilizado para aplicações distribuídas. Sua forma de funcionamente é simples e permite que os processos distribuídos sejam coordenados por meio de um espaço

de nomes (*namespace*) hierárquico e compartilhado, e sua organização é similar a de um sistema de arquivos padrão.

Conforme mostrado na Figura 4.1, o ZooKeeper é composto por diversos servidores. O ZooKeeper possui um algoritmo de eleição que faz com que um dos servidores seja escolhido o líder, e o outros passam a ser chamados de seguidores (*followers*). Caso o líder desconecte-se do ZooKeeper, por exemplo, por uma possível queda do servidor, o algoritmo de eleição é executado novamente para que um dos servidores restantes se torne o novo líder [29].

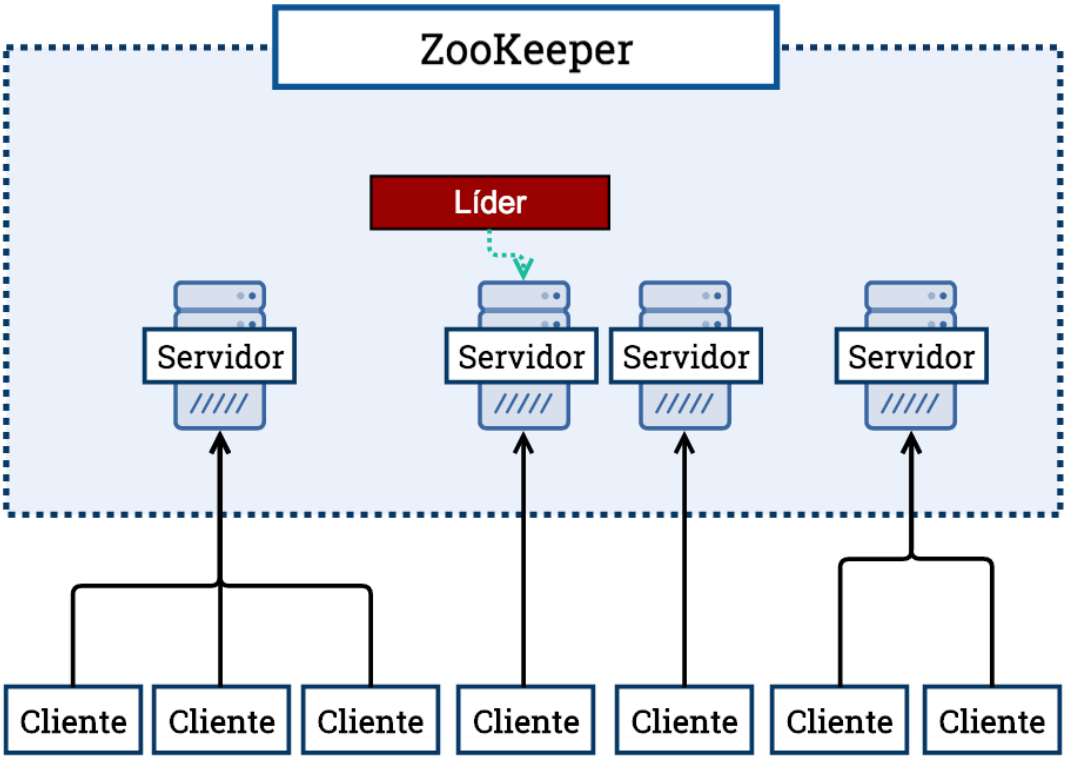

Figura 4.1: Modelo de Serviço Utilizado no ZooKeeper, adaptado de [29].

Com relação aos dados mantidos no ZooKeeper, eles são dispostos em estruturas chamadas ***znodes***, e estes podem ser comparados a arquivos e diretórios dentro de um sistema de arquivos tradicional.

Ao contrário de um sistema de arquivo comum, que é projetado para o armazenamento dos dados, as informações gravadas no ZooKeeper são pequenas (armazenamento máximo limitado a 1 Megabyte) e são mantidos na memória, o que faz com que ele atinja uma baixa latência e altas taxas de transferência. Assim como os servidores coordenados pelo ZooKeeper são replicados para garantir a tolerância a falhas, seu próprio serviço também é replicado em diversas máquinas, a fim de se manter sempre ativo, independente de quedas e falhas [29].

As máquinas coordenadas (clientes) conectam-se a um único servidor ZooKeeper. O cliente mantém uma conexão TCP na qual envia requisições, recebe respostas e envia *heartbeats*, que são notificações periódicas enviadas do cliente ao servidor para avisá-lo que está *online*, e, caso o tempo de resposta ultrapasse um determinado *timeout* (configurado como 2 segundos no ZooKeeper), o servidor assume que o cliente está *offline*. No caso de queda da conexão TCP entre o cliente e o servidor, o cliente tentará se conectar a outro servidor.

O espaço de nomes implementado pelo ZooKeeper é muito parecido com o de um sistema de arquivos padrão, ou seja, um caminho absoluto é uma sequência de caminhos relativos separados por uma barra (/). Cada nó do espaço de nomes do ZooKeeper é identicado por um caminho. A Figura 4.2 mostra um exemplo de uma possível estrutura hierárquica de *znodes* do ZooKeeper.

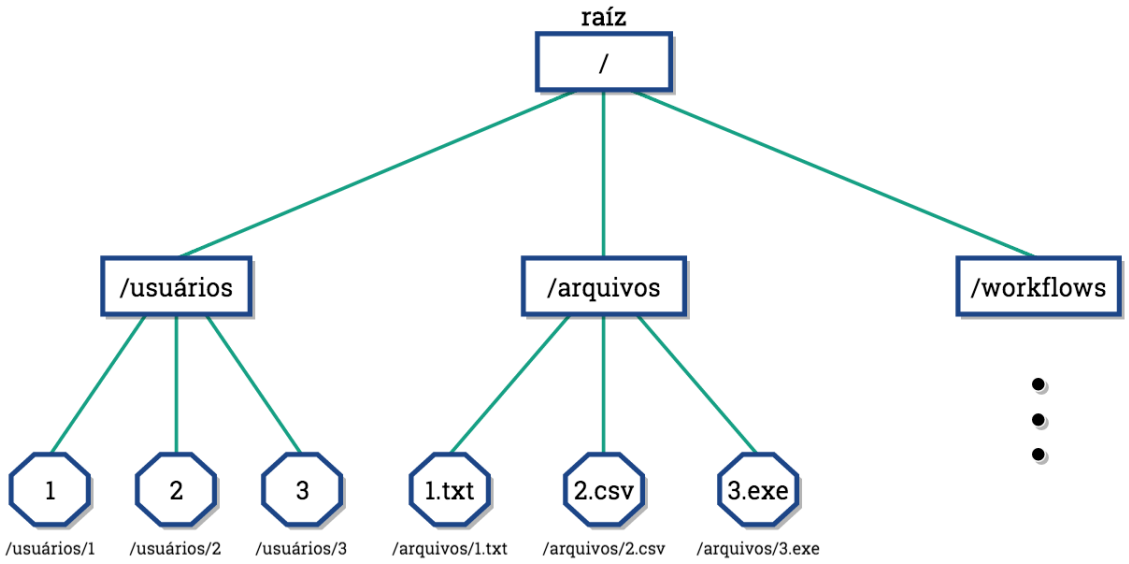

Figura 4.2: Exemplo de Estrutura de Nós do ZooKeeper.

Com relação à notificação de eventos aos clientes, o ZooKeeper implementa o conceito de ***watchers***. Um *watcher* é um aviso e serve para notificar alguma entidade do sistema que um *znode* foi alterado. Dessa forma, para que essa entidade possa notar uma atualização em um *znode*, é necessário adicionar um *watcher* a ele. Assim, dada uma alteração em um nó, apenas quem inseriu um *watcher* nele será notificado [29].

Assim sendo, soma-se às características apresentadas, o fato de o mesmo apresentar uma interface de programação (*API - Application Programming Interface*) muito compacta, suportando operações simples, como:

- ***create*:** Cria um *znode* na estrutura do ZooKeeper;
- ***delete*:** Exclui um *znode*;

- ***getData*:** Obtém os dados de um *znode*;
- ***setData*:** Grava dados em um *znode*;
- ***getChildren*:** Recupera uma lista contendo os nós filhos de um determinado *znode*;
- ***exists*:** Verifica a existência de um nó, dado um caminho.

Devido às características apresentadas, o ZooKeeper então foi escolhido para coordenar os serviços e processos da plataforma BioNimbuZ. Na Seção 4.2 será apresentada a arquitetura do BioNimbuZ anterior ao desenvolvimento do sistema gerenciador de *workflows* científicos, objetivo do presente trabalho.

4.2 Arquitetura do BioNimbuZ

A arquitetura apresentada nesta Seção representa a antiga arquitetura do BioNimbuZ. Uma nova arquitetura foi proposta no presente trabalho e será apresentada e descrita no Capítulo 5.

O BioNimbuZ utilizava uma arquitetura hierárquica, conforme mostrado na Figura 4.3 e possuía três camadas principais: Interface, Núcleo e Infraestrutura, todas gerenciadas pelo ZooKeeper. Suas principais responsabilidades eram as seguintes:

- **Camada de Interface com o Usuário:** Seu principal objetivo era controlar a interação com o usuário, recebendo e processando seus comandos (utilizando um *terminal*), e repassando-os ao núcleo para que fossem devidamente executados;
- **Camada de Núcleo:** Responsável por gerenciar todos os serviços presentes na plataforma (os quais serão descritos na Seção 4.2.2), e também por gerir toda a federação de nuvens utilizada pelo BioNimbuZ.
- **Camada de Infraestrutura:** Sua principal responsabilidade é prover uma interface de comunicação entre o BioNimbuZ e os provedores de nuvens. Consiste em todos os recursos computacionais que os provedores colocam à disposição da federação somados aos seus respectivos *plugins* de integração.

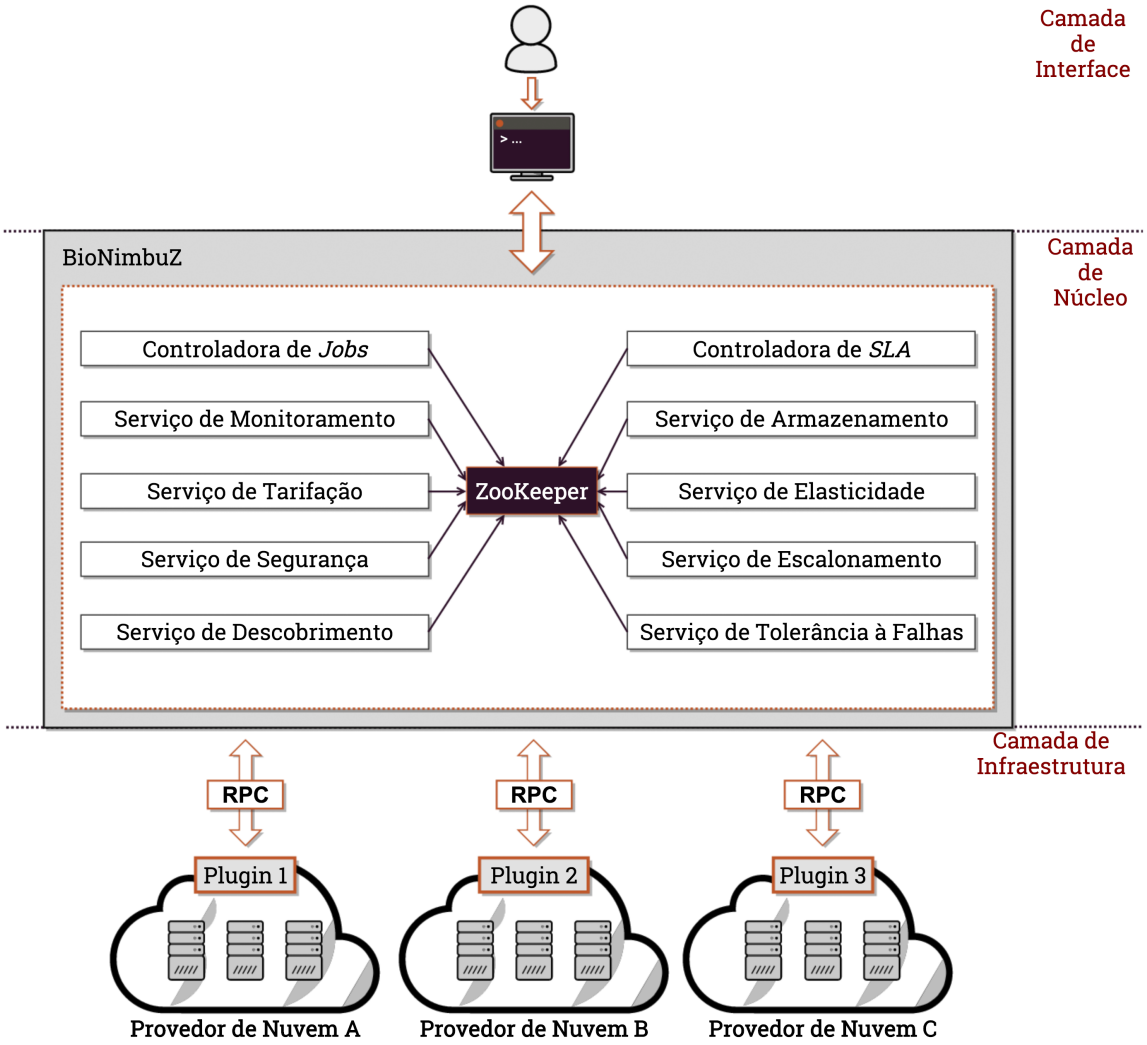

Figura 4.3: Antiga Arquitetura do BioNimbuZ.

As Seções 4.2.1, 4.2.2 e 4.2.3 descrevem os detalhes específicos das três camadas apresentadas na Figura 4.3.

4.2.1 Camada de Interface com o Usuário

Camada responsável por interagir com o usuário, recebendo seus comandos e devolvendo o resultado do processamento realizado pelo núcleo, podendo ser desenvolvida de diversas maneiras: páginas *web*, linhas de comando, interfaces gráficas, sistemas gerenciadores de *workflows*, etc. Para tal, esses serviços de interação precisam se conectar à Internet e se comunicar com a camada abaixo da arquitetura, o núcleo do BioNimbuZ.

Os comandos enviados para o núcleo podem realizar uma série de tarefas, tais como: listar os arquivos armazenados na federação, realizar o *upload* de arquivos, submeter um

job para ser executado, etc. Assim, nota-se que essa camada é a responsável por mostrar ao usuário uma forma de se comunicar com o BioNimbuZ.

Todavia, essa versão do BioNimbuZ tinha implementado apenas a interface via linha de comando. A interface via *web* foi implementada a partir deste trabalho, e será descrita no próximo capítulo.

4.2.2 Camada de Núcleo

Esta camada tem a atribuição de integrar os serviços providos pelo BioNimbuZ, servir como núcleo processador dos comandos provenientes da camada de interface com o usuário, (descrita na Seção 4.2.1), e gerenciar a infraestrutura de federação de nuvens, que é a camada de infraestrutura (a ser descrita na Seção 4.2.3).

O chamado Núcleo do BioNimbuZ possuía oito serviços principais: serviço de descobrimento, serviço de monitoramento, serviço de tolerância a falhas, serviço de escalonamento, serviço de segurança, serviço de armazenamento, serviço de elasticidade e serviço de tarifação. Além destes, possuía mais duas controladoras: controladora de *jobs* e controlador de *SLA*. A fim de exercerem suas funcionalidades, os serviços e as controladoras interagiam entre si por meio do gerenciamento de troca de mensangens provido pelo ZooKeeper. No que tange aos serviços e controladoras, segue uma descrição de suas respectivas funções:

- **Controladora de *Jobs***: É responsável por fazer a ligação entre a camada de interface com o usuário e o núcleo do BioNimbuZ. Ela realiza o controle de acesso dos usuários ao submeter *jobs* aos recursos disponíveis na federação controlada pelo BioNimbuZ. Para realizar a verificação das credenciais do usuário, a Controladora de *Jobs* acessa o Serviço de Segurança. Além disso, a Controladora de *Jobs* é responsável por gerenciar os pedidos dos vários usuários, de forma a fazer o controle por usuário, mantendo os resultados para posterior consulta.

- **Controladora de SLA**: Atualmente, há um esforço muito grande por parte dos provedores de nuvens computacionais para que a qualidade dos serviços (*Quality of Service*) prestados percebidos pelo usuário sejam os melhores possíveis. E isso só é possível a partir de Acordos de Níveis de Serviço (*Service Level Agreement*) assinados entre os usuários e os provedores.

  Nesse contexto, o BioNimbuZ também tem essa preocupação com a qualidade dos serviços prestados aos seus usuários. Assim, quando um usuário submete um *job* para ser executado pela federação do BioNimbuZ, por meio da interface com o usuário, ele deve preencher um *template* de SLA, que representa, entre outras coisas, os parâmetros de *QoS* desejados pelo usuário. Estes parâmetros podem descrever

desde requisitos funcionais, como número de núcleos de CPU, tamanho de memória, tamanho de armazenamento, até requisitos não funcionais, como custo a pagar e taxa de transferência.

Portanto, a Controladora de SLA tem a responsabilidade de verificar se os requisitos especificados no *template* preenchido pelo usuário podem ser suportados pela federação de nuvens naquele dado momento e, para isso, a controladora utiliza os dados de SLA informados pelo *plugin* de integracão de cada provedor.

- **Serviço de Tolerância a Falhas**: Em ambientes de computacão em nuvem, falhas em máquinas são fatos comuns e ocorrem com frequência. Logo, uma federação de nuvens deve sempre levar em consideracão requisitos de tolerância a falhas e desenvolvimento de ambientes de alta disponibilidade.

  O Serviço de Tolerância a Falhas tem como objetivo principal garantir que todos os serviços do BioNimbuZ estejam sempre disponíveis e, em caso de falhas, inicie alguma ação de recuperação. Essa recuperação exige que todos os componentes do sistema que foram afetados pela falha voltem ao seu estado anterior. O serviço de tolerância a falhas deve atuar de forma distribuída na plataforma, e estar presente em outros serviços, monitorando seu estado.

  No Serviço de Armazenamento, por exemplo, quando um alerta de indisponibilidade de um recurso é lançado por um *watcher*, é iniciada uma rotina de recuperação para os arquivos que este recurso continha. Como os arquivos são replicados na federação, a recuperação ocorre de forma a identificar quais réplicas foram perdidas, realizando uma nova duplicação em outro servidor do BioNimbuZ. A tolerância a falhas no BioNimbuZ está intimamente ligada aos *watchers* do ZooKeeper, pois são estes que disparam os alertas de indisponibilidade, notificando o sistema sobre problemas em algum recurso da federação.

- **Serviço de Escalonamento**: É o serviço responsável por receber o pedido de execução de algum *job* e distribuí-lo dinamicamente para os provedores disponíveis, dividindo-os em instâncias menores - chamadas *tasks*. O serviço de escalonamento também é responsável por acompanhar toda a execução de um *job*, e manter um registro das execuções já escalonadas.

  Para realizar a distribuição de tarefas na federação, algumas métricas são levadas em consideração, tais como latência, balanceamento de carga, tempo de espera e capacidade de processamento dos recursos disponíveis, visando atender o que foi determinado no acordo de SLA.

Atualmente, o BioNimbuZ tem implementado três políticas de escalonamento, cada uma focando em um objetivo diferente [15] [14] [4].

- **Serviço de Segurança**: Segurança em nuvens federadas é uma área de estudo em constante evolução, e este serviço deve trabalhar em diversos pontos para garantir uma execução efetivamente segura das tarefas, mitigando possíveis pontos de falhas no sistema.

  Outros pontos constituem o cerne deste serviço: autenticação, autorização, confiabilidade e integridade. O primeiro, está focado em saber se o usuário que está tentando acessar algum recurso na federação é quem ele realmente diz ser. Depois de estar autenticado no sistema, o segundo passo é a autorização, que consiste em verificar se o usuário pode realizar aquela ação que está demandando.

  Muitos outros aspectos podem ser abordados por este serviço, como a criptografia de mensagens, para garantir a confidencialidade na troca de informações entre provedores, por exemplo, e também a verificação de integridade de arquivos, de modo que seja possível garantir que um arquivo não seja alterado por fatores externos à federação, como por exemplo um componente de rede gerando erros nos dados trafegados.

- **Serviço de Armazenamento**: Este serviço é o responsável pela estratégia de armazenamento dos arquivos que são utilizados ou mantidos pelo BioNimbuZ. O serviço deve decidir sobre a distribuição dos arquivos entres os diferentes provedores da federação. Para isso, o serviço de armazenamento comunica-se com o Serviço de Descobrimento para obter acesso as informações sobre a federação. Com isso, o serviço saberá as condições atuais de armazenamento de cada um dos provedores que faz parte da federação.

  O armazenamento deve ocorrer de forma eficiente, para que as aplicações possam utilizar os arquivos com o menor custo possível. Este custo é calculado utilizando-se algumas métricas, tais como latência de rede, distância entre os provedores e capacidade de armazenamento. Também é adotada a replicação dos arquivos em mais de um provedor, visando a diminuição de custos envolvidos.

  O serviço de armazenamento do BioNimbuZ possui políticas de armazenamento implementadas, uma delas, pode ser vista em [13].

- **Serviço de Elasticidade**: A função do serviço de elasticidade é, dinamicamente, aumentar ou diminuir o poder computacional total da federação. Para tal, gera novas instâncias ou desliga instâncias ociosas de máquinas virtuais. Além disso, ele também reajusta as configurações de CPU, de memória, de largura de banda,

entre outros recursos que são disponibilizados pelos provedores de nuvem, a fim de se obter uma melhor utilização da infraestrutura disponível.

A elasticidade pode ser vertical, quando há um redimensionamento das configurações da máquina virtual, como também pode ser horizontal, quando é alterado o número de máquinas virtuais instanciadas nos provedores.

- **Serviço de Tarifação**: Tem a responsabilidade de calcular o quanto os usuários devem pagar pela utilização dos serviços oferecidos na plataforma BioNimbuZ. Para que isto seja possível, este serviço se mantém em constante contato com o serviço de monitoramento, para obter informações, tais como tempo de execução e quantidade de máquinas virtuais alocadas para as tarefas que foram executadas por determinado usuário. Dessa forma, é possível verificar quais recursos foram alocados para determinada carga de trabalho submetida pelo usuário, calculando assim, o preço devido por cada usuário da plataforma.

- **Serviço de Descobrimento**: Responsável por identificar e manter informações tais como poder computacional, capacidade de armazenamento e latência de rede dos provedores de nuvem presentes na federação. Também mantém detalhes sobre parâmetros de execução e arquivos de entrada e de saída dos softwares disponíveis para execução pelo usuário.

  Toda vez que um provedor é incluído na federação de nuvens, seus dados são gravados em um novo *znode* no ZooKeeper. Assim, para qualquer outro serviço obter informações acerca dos provedores presentes na federação, basta realizar uma consulta simples no ZooKeeper. Sempre que as informações acerca de um provedor são atualizadas (por exemplo, com a diminuição do poder computacional ou falta de espaço de armazenamento), outros serviços são notificados, a partir dos *watchers* que foram adicionados. Dessa forma, todo o sistema tem acesso a essas informações.

- **Serviço de Monitoramento**: Este serviço monitora a nuvem federada verificando as aplicações e os *jobs* para a execução. Ao receber um pedido de execução de um *job*, vindo da Controladora de *Jobs*, o Serviço de Monitoramento identifica se o software computacional relacionado àquele *job* está disponível em algum provedor de serviço, e então redireciona o pedido para o Serviço de Escalonamento, o qual processa o pedido garantindo que todas as requisições sejam devidamente atendidas e executadas.

  A fim de retribuir informações acerca da execução das tarefas, mensagens periódicas são enviadas para os recursos gerenciados pelo BioNimbuZ. Além disso, notifica a Controladora de *Jobs* sobre o estado da execução de determinado *Job*.

Dessa forma, é por meio dessas mensagens periódicas que o Serviço de Monitoramento é capaz de vericar se uma execução é finalizada com sucesso sem violar os parâmetros de SLA acordados com o usuário.

4.2.3 Camada de Infraestrutura

A camada de infraestrutura consiste nos recursos computacionais disponibilizados na federação por meio dos provedores de nuvem, visíveis ao BioNimbuZ através dos *plugins* de integração.

Essa camada provê meios para que haja a comunicação entre o BioNimbuZ e os provedores presentes na federação, mapeando os comandos vindos da arquitetura do BioNimbuZ para comandos específicos de cada provedor e vice-versa.

4.3 Considerações Finais

Esse capítulo abordou os detalhes da plataforma de federação de nuvens BioNimbuz, apresentando os serviços e controladoras que os compõe. Também foram mostrados detalhes sobre as tecnologias Apache Avro e Apache ZooKeeper, ambas utilizadas no BioNimbuZ.

O Capítulo 5 apresentará os passos do desenvolvimento do sistema gerenciador de *workflows* proposto, e também traz as soluções projetadas para tratar os problemas de comunicação, persistência dos dados e controle de acesso à plataforma BioNimbuZ.

Capítulo 5

Sistema Gerenciador de *Workflows* Científicos para o BioNimbuZ

Este capítulo trata das etapas do desenvolvimento do Sistema Gerenciador de *Workflows* Científicos para a plataforma BioNimbuZ, e as alterações necessárias para que ele fosse suportado pelo atual estado do sistema. Primeiramente, na Seção 5.1 é apresentado o funcionamento do BioNimbuZ antes do desenvolvimento do sistema gerenciador de *workflows* proposto neste trabalho, o qual era executado através de um *terminal*. Os novos requisitos do sistema são apresentados na Seção 5.2. Com esses requisitos, foi proposta uma nova arquitetura para o sistema, a qual é apresentada na Seção 5.3. A Seção 5.4 mostra os detalhes de como a Aplicação *Web* foi desenvolvida para dar suporte ao Sistema Gerenciador de *Workflows*. Na Seção 5.5 são expostos os motivos da construção de um novo componente da arquitetura de software, chamada Camada de Integração, responsável por realizar a troca de mensagens entre a Camada de Núcleo do BioNimbuZ e a Camada de Aplicação utilizando *webservices REST*. Por último, na Seção 5.6 são apresentadas as alterações feitas no BioNimbuZ para suportar a persistência de dados.

5.1 Do *terminal* para a Interface *Web*

O BioNimbuZ primeiramente projetado e implementado por Hugo Saldanha [68] era acessado via *terminal*. Nele, o usuário digitava uma série de comandos, os quais eram processados e executados pelo servidor do BioNimbuZ, e então recebia de volta o resultado daquele comando. Essa forma não era intuitiva e nem amigável para o usuário, pois exigia que o mesmo conhecesse a execução de comandos através de um *terminal*.

Assim, por exemplo, para submeter uma sequência de passos, ou *jobs*, formando assim um *workflow*, o usuário deveria executar diversos comandos em sequência, ou seja, o usuário era quem gerenciava manualmente todos os passos de seu *workflow*. Outro ponto

negativo dessa abordagem era o fato de que o usuário não poderia acessar o sistema, ver os *jobs* submetidos ou visualizar seus arquivos de qualquer lugar, pois era necessário realizar o *download* do executável do BioNimbuZ em sua máquina local. Uma das vantagens da nova abordagem de sistema, baseada em tecnologias *web*, é a acessibilidade provida pela *Internet*. Dessa forma, o usuário pode, por exemplo, iniciar um *workflow* em seu laboratório, concluí-lo e submetê-lo em casa e verificar sua execução pelo celular.

Na abordagem anterior, baseada em *terminal*, era necessário realizar o *download* do *.JAR* (*Java Archive*) do projeto e executá-lo via linha de comando. A Figura 5.1 mostra a interface percebida pelo usuário ao executar o BioNimbuZ no sistema operacional Linux.

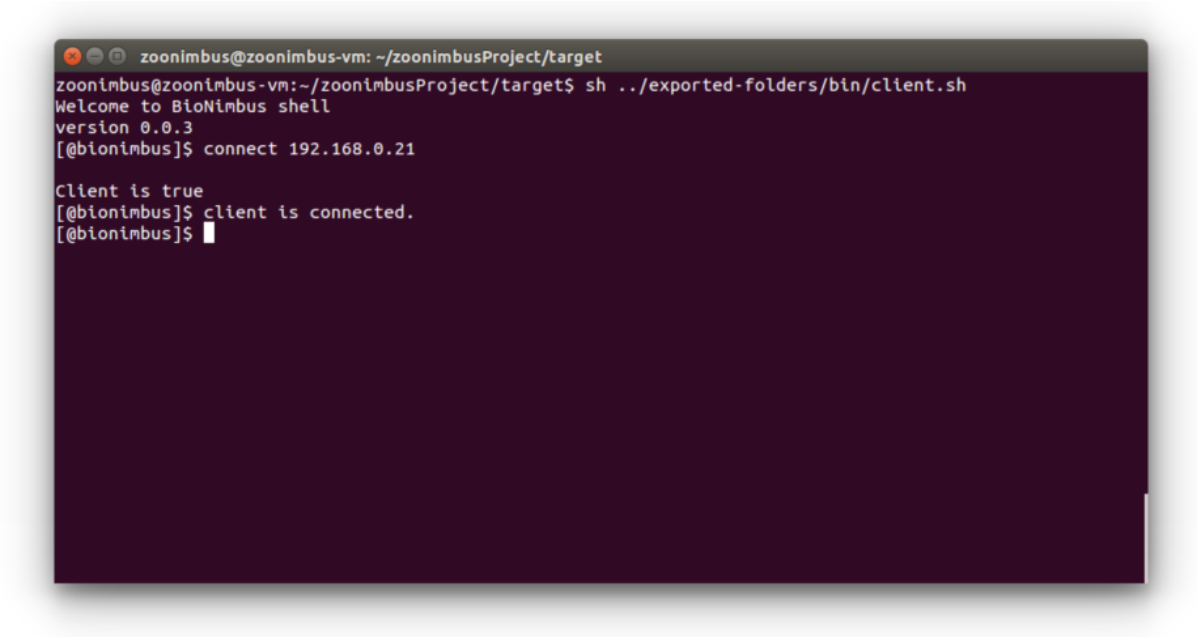

Figura 5.1: *Terminal* do BioNimbuZ.

Dessa forma, utilizando a interface mostrada na Figura 5.1, para o usuário submeter um *workflow* com dois passos, a seguinte sequência de comandos deveria ser executada:

1. Executar o comando ***connect*** para se conectar ao BioNimbuZ, passando o IP da máquina servidora como argumento do comando;
2. Enviar cada arquivo a partir do comando ***upload***, passando o caminho do arquivo a ser enviado;
3. Com todos os arquivos enviados, submeter o comando ***start*** passando o identificador do primeiro serviço (nesse caso, serviço se refere ao software computacional a ser executado naquele *job* e uma lista contendo os serviços disponíveis era mostrada a partir do comando ***services***), a lista de arquivos de entrada e a lista de arquivos de saída;

4. A partir do início de um *job* era possível verificar seu *status* com o comando ***status***, informando o número do *job*.

Os comandos apresentados descrevem o passo a passo da execução de apenas um *job*. Para um segundo *job*, seria necessário submeter todos esses comandos novamente, com os dados de entrada requeridos pelo segundo *job*. Assim, nota-se que o método descrito, via *terminal*, não dava suporte à execução de *workflows*, pois não era possível ligar elementos, formar dependências e controlar o fluxo de dados de maneira única. Eram necessárias diversas iterações da sequência de comandos descrita para se ter uma ligação entre os *jobs* que o usuário quisesse executar. Dessa forma, o usuário deveria ter informações que, nem sempre, eram de seu conhecimento, como o *IP* da máquina servidora do BioNimbuZ, o caminho do arquivo ou os argumentos de um serviço provido.

A próxima seção apresenta os requisitos que foram verificados e implementados, com o objetivo de se desenvolver o Sistema Gerenciador de *Workflows* Científicos proposto no presente trabalho.

5.2 Novos Requisitos

De acordo com as deficiências notadas no BioNimbuZ, verificou-se que o acesso aos serviços providos pelo BioNimbuZ, via *terminal*, poderia ser melhorado caso alguns requisitos fossem implementados, tais como:

- **Acesso seguro pela *Internet*:** Prover um meio de acesso seguro a plataforma do BioNimbuZ à qualquer pessoa conectada à *Internet*;
- **Composição do *Workflow* de Maneira Gráfica:** Um *workflow* científico é composto por uma sequência de passos interligados, formando uma cadeia de execução com entradas e saídas. A forma mais viável de projetá-lo seria a partir de uma interface que possibilitasse o desenho de um *workflow*;
- **Acesso à Plataforma Utilizando-se Usuário e Senha:** O acesso via *terminal* não dava suporte a múltiplos usuários. Cada usuário deveria ter o BioNimbuZ instalado em seu computador e submeter sua sequência de *jobs*, da sua máquina para o servidor do BioNimbuZ. Assim, por questões de segurança, fez-se necessário o desenvolvimento de um método de controle de usuários, verificando a autenticidade e a autorização no momento do *login*;
- **Facilidade no Envio dos Arquivos:** O método de envio dos arquivos poderia ser facilitado, caso fosse desenvolvido uma maneira gráfica de enviá-los ao BioNimbuZ;

- **Controle do *Status* dos *Workflows* Submetidos:** Método de visualização do *status* de cada *workflow* submetido para ser processado pela plataforma do BioNimbuZ;
- **Separação entre Aplicação Cliente e Aplicação Servidora:** Evitaria um ponto único de falhas, pois as aplicações poderiam ser executadas em ambientes diferentes, em infraestruturas diferentes e, no caso de falha de uma delas, a outra não seria prejudicada. Com esse requisito também seria possível reduzir custos, pois a infraestrutura necessária para executar a aplicação cliente não tem a necessidade de ser tão robusta quanto a aplicação servidora, pois apenas esta executaria os serviços computacionais providos pelo BioNimbuZ;
- **Persistência dos Dados:** Prover formas para que informações, tais como *status* dos *workflows* de um usuário e histórico de execução, fossem persistidas em bancos de dados, para acesso posterior pelo usuário.

Na próxima seção será apresentada a nova arquitetura do BioNimbuZ, contendo os novos componentes de software e seu detalhamento.

5.3 Proposta de uma Nova Arquitetura de Implementação para o BioNimbuZ

Tendo em vista os requisitos levantados, descritos na Seção 5.1, e para que o BioNimbuZ se tornasse acessível através da Internet, foi necessária a implementação de um método de acesso baseado em tecnologias *web*, substituindo o método antigo via *terminal*. Para isso, foi desenvolvido um sistema gerenciador de *workflows* com interface gráfica, utilizando a linguagem de programação Java em conjunto com *frameworks web*. Para tal, uma nova proposta de arquitetura de implementação foi concebida, englobando as modificações necessárias, principalmente, na parte de comunicação entre os componentes do sistema: Aplicação *Web* e Núcleo.

Nessa nova arquitetura, o BioNimbuZ não será composto apenas pelas três camadas projetadas anteriormente (Aplicação, Núcleo e Infraestrutura [68]). Será acrescentado um novo componente chamado Camada de Integração, devido à importância da correta troca de mensagens entre a Camada de Aplicação e a Camada de Núcleo do BioNimbuZ. Assim, a Figura 5.2 apresenta essa nova proposta de arquitetura, e uma breve descrição dos componentes:

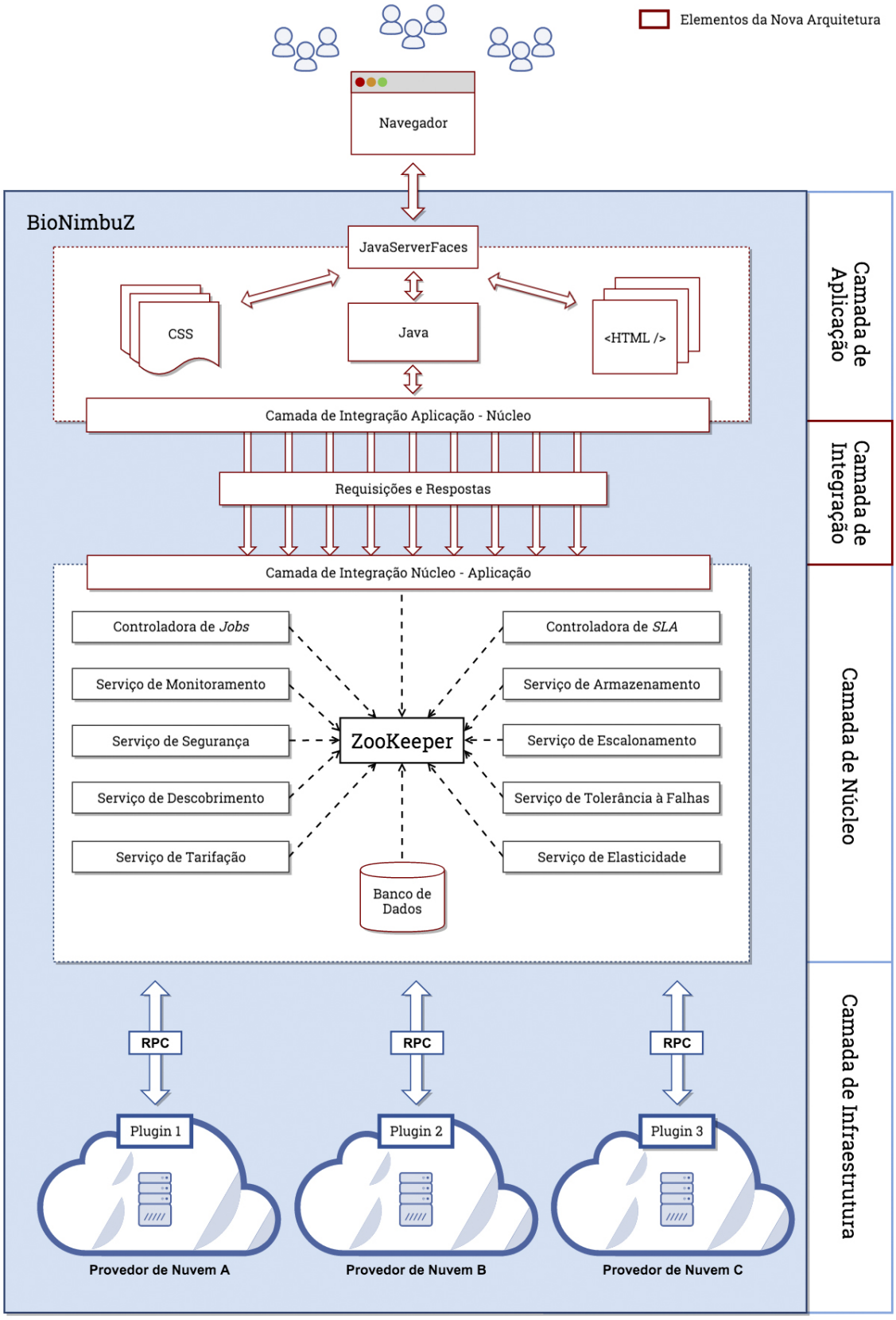

Figura 5.2: Nova Arquitetura do BioNimbuZ.

- **Camada de Aplicação:** É composta pela aplicação *web* e implementa o sistema gerenciador de *workflows* científicos e utiliza a Camada de Integração para enviar requisições e receber respostas da Camada de Núcleo. Anteriormente, na arquitetura do BioNimbuZ, essa camada era denominada Camada de Interface, pois apenas realizava a interação com o usuário. Com o desenvolvimento da aplicação *web* proposta nesse trabalho, essa camada passa a ser chamada de Camada de Aplicação;
- **Camada de Integração:** Responsável por integrar as Camadas de Aplicação e Núcleo. Realiza a troca de mensagens entre essas camadas utilizando *webservices*, disparando requisições da Camada de Aplicação para o Núcleo e recebendo respostas do Núcleo para a Aplicação;
- **Camada de Núcleo:** Responsável pelo armazenamento de arquivos, escalonamento de *jobs*, execução de *workflows*, descobrimento dos provedores e o gerenciamento de possíveis falhas. Nessa camada (existente na arquitetura prévia do BioNimbuZ) será incluído o componente banco de dados, responsável pela persistência dos metadados necessários para visualização das informações pela camada de aplicação;
- **Infraestrutura:** Composta pelos provedores de serviço utilizados pelo BioNimbuZ e que compõem sua federação de nuvens. Essa nova proposta de arquitetura não engloba alterações nessa camada.

A Seção 5.4 traz as modificações realizadas na Camada de Aplicação com a implementação da aplicação *web* e a maneira a qual foi concebida.

5.4 Camada de Aplicação

Esse componente compreende o Sistema Gerenciador de *Workflows* Científicos e deve prover meios para que os usuários possam logar no sistema BioNimbuZ através de uma interface gráfica acessível pela Internet, e utilizem suas funcionalidades, como: envio de arquivos, que serão utilizados como entrada dos *workflows* criados pelo usuário; exclusão de arquivos; visualização dos dados de saída gerados pela execução de seus *workflows*, por exemplo.

Pela interface, o usuário também deve ser capaz de criar e projetar seus *workflows* de maneira gráfica, ligando passos, indicando dependências, incluindo argumentos e indicando quais arquivos de entrada serão utilizados.

Com os requisitos descritos na Seção 5.2, foi projetada a implementação desse componente através da linguagem de programação Java e *frameworks Web*, que são bibliotecas

de software que facilitam a implementação de funcionalidade relacionadas aos sistemas voltados para *Internet*.

Na implementação de uma aplicação *Web*, o desenvolvimento é dividido em camadas e utiliza-se, normalmente, o padrão de projeto *MVC* (*Model-View-Controller*) [32]. O padrão *MVC* é utilizado no desenvolvimento de interfaces pela sua característica de estar intimamente mapeado nos aspectos principais de uma interface, que são [32]:

- **Visualização:** Fornece a apresentação dos dados presentes no Modelo;
- **Controlador:** Recebe os dados e comandos do usuário, e determina o que isso significa para o Modelo;
- **Modelo:** Contém todos os dados e estados persistidos, geralmente, em bancos de dados.

Dessa forma, o padrão *MVC* contém exatamente esse três componentes: o ***Model*** (Modelo), o ***Controller*** (Controlador) e o ***View*** (Visualização). A interação entre esses componentes pode ser visualizada na Figura 5.3 e descrita conforme os seguintes passos:

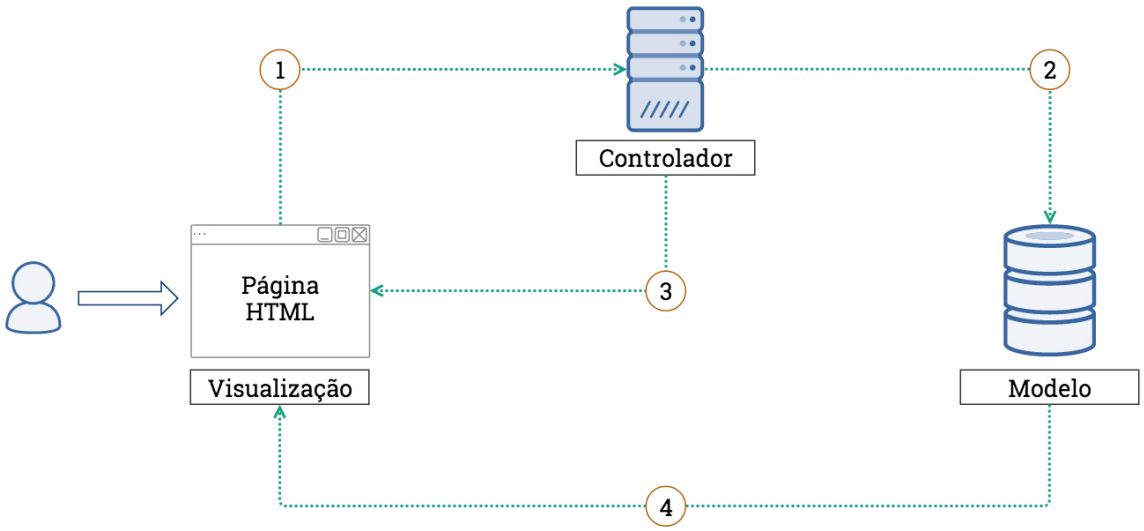

Figura 5.3: Modelo *Model-View-Controller*, adaptado de [32].

- **Passo 1:** O usuário realiza uma ação na página HTML;
- **Passo 2:** O Controlador processa o comando e muda o estado de algum objeto no Modelo;
- **Passo 3:** O Controlador envia um comando à Visualização para modificar os dados exibidos;
- **Passo 4:** O Modelo notifica a Visualização de que os dados foram modificados;

Nesse contexto, diversos *frameworks* surgiram para o desenvolvimento de aplicações *web*, tais como: *Spring MVC* [75], *Struts* [31], *JSF* (*JavaServerFaces*) [58], *Play* [60], entre outros. Para este projeto, o *framework* escolhido foi o *JavaServerFaces*, pois ele é uma especificação Java para a construção de interfaces de usuário baseadas em componentes para aplicações *web*. Além disso, ele possui um modelo de programação dirigido a eventos, abstraindo os detalhes da manipulação e da organização dos componentes, permitindo que o programador se concentre na lógica da aplicação.

Dessa forma, utilizando-se o padrão MVC, descrito anteriormente, foi projetada e implementada a aplicação *web* contendo os componentes necessários para o desenvolvimento do sistema gerenciador de *workflows* científicos, os quais serão descritos nas próximas seções.

5.4.1 Camada de Modelo

Geralmente, a primeira fase do desenvolvimento de um software é o levantantamento dos requisitos e definição das entidades principais do sistema, formando assim um modelo do sistema. As entidades de um sistema podem ser qualquer coisa que façam sentido na solução do problema específico, para o qual aquele sistema foi criado. Por exemplo, um usuário do mundo real pode ser mapeado em uma entidade de sistema chamada *User*, com uma lista de características, tais como Nome, Cadastro de Pessoa Física, Registro Geral, Sexo, Telefone, Email, etc. A depender do sistema, aspectos específicos podem ser definidos também, por exemplo, em um sistema médico, atributos como tipo sanguíneo e densidade óssea são importantes, mas ao mesmo tempo, esses atributos, geralmente, não tem finalidade em um sistema bancário, por exemplo.

Assim, essa camada deve prover entidades mapeadas que façam sentido para o sistema (com atributos bem definidos), e também deve garantir que seu estado seja persistente e íntegro a cada inclusão, atualização ou remoção.

Nesse cenário, foram mapeadas entidades importantes para o sistema BioNimbuZ. Também foi desenvolvido o acesso a banco de dados para garantir a persistência dessas informações e, para isso, o sistema gerenciador de banco de dados escolhido foi o **MySQL** [59], pois ele é confiável, de fácil instalação e manutenção, e por possibilitar a replicação dos dados, evitando assim um ponto único de falhas. São armazenados nesse banco de dados apenas metadados (informações que representam características do dado real) para auxiliar a visualização das informações na camada de aplicação. Assim, não é necessário um banco de dados mais robusto para persistir grandes dados e volumes (como informações sobre sequeciamentos genéticos e grandes arquivos de referência, como o genoma humano), pois estes devem ser persistidos em ferramentas de grande armazenamento, como a Amazon S3 [72].

Além disso, foi definida que sua localização e seu método de acesso deveriam ser desenvolvidos no núcleo, conforme proposta de nova arquitetura apresentada na Seção 5.3. Os detalhes do método de acesso à base de dados desenvolvida serão retratados na Seção 5.6.

Assim, as principais entidades do sistema e seus respectivos atributos são descritos nas Tabelas 5.1, 5.2, 5.3 e 5.4 :

- **`User`:** Contém todos os dados necessários para identificar um usuário no sistema.

Tabela 5.1: Lista de Atributos da Entidade `Usuário`.

Atributo	Descrição
`id`	Sequência de caracteres que identifica unicamente um usuário no sistema.
`login`	Contém o *login* para o usuário entrar no sistema.
`password`	Senha utilizada para entrar no sistema.
`name`	Contém o nome do usuário.
`cpf`	Cadastro de Pessoa Física do usuário.
`cellphone`	Celular do usuário.
`storageUsage`	Contém a soma de *bytes* dos arquivos enviados pelo usuário ao sistema.
`List<FileInfo>`	Contém a lista de arquivos pertencente ao usuário.
`List<Workflows`	Lista de *workflows* submetidos pelo usuário.

- **`FileInfo`:** Contém as informações acerca de arquivos que são enviados à plataforma BioNimbuZ.

Tabela 5.2: Lista de Atributos da Entidade `FileInfo`.

Atributo	Descrição
`id`	Identificador único do arquivo.
`name`	Nome do arquivo.
`size`	Tamanho em *bytes* do arquivo.
`userId`	Identificador do usuário que realizou o *upload* deste arquivo.
`uploadTimestamp`	Dia e horário do *upload* do arquivo.
`hash`	Contém o *hash* do arquivo para verificação de sua integridade.
`payload`	*Bytes* contendo o arquivo em si.

- **`Job`:** Define um passo do *workflow*. No BioNimbuZ, todo *workflow* é composto por uma sequência de *jobs*.

Tabela 5.3: Lista de Atributos da Entidade `Job`.

Atributo	Descrição
`id`	Identificado do *job*.
`serviceId`	Identificador do software computacional disponibilizado pelo BioNimbuZ.
`args`	Campo opcional que contém os argumentos de entrada de um dado software.
`List<FileInfo>`	Lista de arquivos de entrada daquele *job*.
`output`	Nome do arquivo de saída deste passo.
`dependencies`	Contém os passos que devem ser executados antes da execução desse *job*.

- **`Workflow`:** Descreve a entidade alvo desse sistema, o *workflow* contendo sua lista de passos (*jobs*).

Tabela 5.4: Lista de Atributos da Entidade `Workflow`.

Atributo	Descrição
`id`	Identificador do *workflow*.
`List<Job>`	Lista de *jobs* que compõem o *workflow*.
`datestamp`	Dia e horário da criação do *workflow*.
`userId`	Identificador do usuário que criou o *workflow*.
`description`	Breve descrição do *workflow*.
`status`	Estado atual do *workflow* (pendente, executando, pausado, com erros, etc).

Assim, com as entidades do sistema projetadas, passou-se ao desenvolvimento da próxima camada do modelo *MVC*, a Camada de Visualização. Seus detalhes são apresentados na Seção, 5.4.2.

5.4.2 Camada de Visualização

Após o mapeamento das entidades necessárias para o sistema gerenciador de *workflow*, foi desenvolvida a camada de visualização do modelo *MVC*. Para tal, foram implementadas páginas em **HTML** (*Hypertext Markup Language*) [21] utilizando componentes visuais da biblioteca *Primefaces* [61]. O *framework* ***Primefaces*** possibilita utilizar componentes visuais atraentes e de maneira simples, sem se preocupar em ajustar parâmetros visuais ou de programação.

Isso significa que, utilizando o *Primefaces*, não é necessário desenvolver os códigos de estilo *CSS* (*Cascading Style Sheets*) [45] os quais definem como os elementos que compõem uma página, um documento ou uma aplicação *web* serão visualmente compostos.

Primefaces também possibilita a criação de páginas *web* sem desenvolver código *JavaScript* [36], que é uma linguagem de programação baseada em *scripts*, utilizada na

parte cliente de páginas *web*, isto é, código executado no navegador do usuário, isso torna o processo de desenvolvimento muito mais eficaz, dado que não é necessário desenvolver a interação entre os elementos. Assim, essas duas características do *Primefaces* facilitam a criação de páginas *web*, possibilitando o desenvolvimento de páginas mais interativas e dinâmicas.

A partir da escolha de tecnologias, foram então desenvolvidas as páginas *web* do sistema. O resultado será apresentado no Capítulo 6, o qual trata dos resultado obtidos com o desenvolvimento do Sistema Gerenciador de *Workflows*. A Tabela 5.5 apresenta a lista de páginas e suas respectivas funções.

Tabela 5.5: Lista de Páginas *HTML* e suas Respectivas Funções no Sistema.

Página *HTML*	**Função**
`login.html`	Página de *login*
`sign_up.html`	Página de cadastro de novos usuários
`file_upload.html`	Página relativa à *upload* de arquivos
`delete_file.html`	Página de deleção de arquivos do usuário
`workflow_composer.html`	Página de composição, *design* e submissão de *workflows*
`workflow_status.html`	Página para visualização do *status* dos *workflows* gerenciados pelo usuário
`workflow_history.html`	Contém o histórico de execução de um determinado *workflow*

A Seção 5.4.3 trata da Camada de Controle e as principais classes envolvidas em seu desenvolvimento.

5.4.3 Camada de Controle

Conforme o que foi exposto previamente, a Camada de Controle tem a responsabilidade de interpretar os dados enviados pelo usuário, a partir da Camada de Visualização (que neste projeto foi desenvolvida a partir de páginas *HTML*), sinalizando ao sistema o que aqueles comandos representam. Essa passagem de dados, da página *HTML* para as classes *Java*, é de responsabilidade do *framework MVC JSF*.

Assim, foram desenvolvidas classes Java controladoras para tratar as requisições disparadas nas páginas *web* de forma que houvesse uma correspondência entre as classes Java e suas respectivas páginas *HTML*. Essa correspondência é mostrada na Tabela 5.6 e descrita a seguir:

Tabela 5.6: Relação entre Classes Java e Páginas *HTML*.

Classe Java	Página Relacionada
`SessionBean`	`login.html`
`SignUpBean`	`sign_up.html`
`FileUploadBean`	`file_upload.html`
`DeleteFileBean`	`delete_file.html`
`WorkflowComposerBean`	`workflow_composer.html`
`WorkflowStatusBean`	`workflow_status.html`
`WorkflowHistoryBean`	`workflow_history.html`

- **`SessionBean`:** Esta classe tem como função principal realizar ações relacionadas à sessão do usuário no sistema, por exemplo executar o *login* e o *logout*. Também deve manter as informações acerca do usuário (dados pessoais, lista de arquivos, lista de *workflows* submetidos, etc) para ser consultada por outros componentes do sistema, funcionando, portanto, como um repositório de dados de usuários;
- **`SignUpBean`:** Controla o fluxo do sistema quando um usuário deseja se cadastrar na plataforma, isto é, verifica se o usuário já foi cadastrado previamente, e, em caso negativo, realiza sua inclusão, persistindo-o no banco de dados;
- **`FileUploadBean`:** Responsável por enviar os arquivos dos usuários à plataforma BioNimbuZ. Também realiza o tratamento do arquivo, verificando aspectos tais como:
  1. **Extensão:** Verifica se a extensão do arquivo satisfaz àquela esperada pelo sistema;
  2. **Tamanho:** Certifica-se de que o tamanho do arquivo enviado não ultrapassa o limite estipulado;
  3. **Nome de Arquivo:** Verifica se aquele nome de arquivo já não foi previamente enviado ao sistema.
- **`DeleteFileBean`:** Sua função é deletar um arquivo de usuário, enviando a requisição de exclusão para a Camada de Núcleo do BioNimbuZ e mostrando o resultado ao usuário (arquivo excluído ou erro no processamento);
- **`WorkflowComposerBean`:** Essa classe é a principal classe do sistema gerenciador de *workflows* desenvolvido. Ela controla a página relativa à composição dos *workflows*, fazendo toda a verificação e a validação dos dados de entradas, dos parâmetro de execução, do *design* do *workflow*, etc;

- **`WorkflowStatusBean`:** Requisita à Camada de Núcleo do BioNimbuZ o estado de um determinado *workflow*, enviando, para isso, seu `ID`. Além disso, formata os dados de retorno, enviando-os à página *HTML* para serem mostrados ao usuário;
- **`WorkflowHistoryBean`:** Envia requisições ao BioNimbuZ pedindo o passo-a-passo da execução de um dado *workflow* (***log***). Com este *log*, o usuário pode, por exemplo, rastrear possíveis erros, propor melhorias ao seu fluxo ou analisar o passo-a-passo de execução de seu *workflow*. Por fim, esta classe formata a resposta enviada pela Camada de Núcleo e a disponibiliza à página *web*.

A listagem acima demonstra a maioria das classes Java desenvolvidas neste projeto com o objetivo de controlar a Camada de Visualização, mas não todas. Outras classes auxiliares também foram implementadas, mas, por possuírem menor importância, não foram aqui citadas.

Dessa forma, essas classes realizam a função de controle do modelo *MVC* que norteia o desenvolvimento da aplicação *web*. A Seção 5.5 trata da solução desenvolvida a fim de tratar o problema de integração das Camadas de Aplicação e Núcleo do BioNimbuZ.

5.5 Camada de Integração

Um dos requisitos da solução proposta era que os componentes de software compostos pela Camada de Aplicação e pela Camada de Núcleo do BioNimbuZ fossem desenvolvidos de maneira separada. Com isso, surgiu o problema de integração entre esses dois elementos.

Nesse contexto, a solução desenvolvida tem como base a comunicação via *webservices REST* (***RE****presentational* ***S****tate* ***T****ransfer*) [20]. *REST* é definido como "*estilo de arquitetura para sistemas de hipermídia distribuídos*" [20]. O *REST* é voltado para sistemas baseados na Internet e tem sido amplamente utilizado na integração de sistemas, pois utiliza operações definidas no protocolo *HTTP* (tais como *PUT*, *GET* e *DELETE*). Assim, softwares que usam o protocolo *HTTP*, podem utilizar, geralmente, soluções baseadas em *REST*.

Dessa forma, para possibilitar a troca de mensagens entre a Camada de Aplicação e a Camada de Núcleo do BioNimbuZ, foi necessário o desenvolvimento de três entidades principais: **Requisições** (*requests*), **Respostas** (*responses*) e **Ações** (*actions*). Estas foram implementadas como `interface` Java, a fim de manter o código o mais genérico possível. Suas responsabilidades são as que seguem:

- **Requisições:** De acordo com Fielding [20], toda requisição feita a um servidor deve conter todo o conjunto de informações necessárias para que possa ser executada corretamente, pois a comunicação não deve manter estado (*stateless*). Assim,

foram desenvolvidas requisições que contivessem todos os dados necessários para a execução daquela ação requisitada pela Camada de Aplicação;

- **Respostas:** Definem a mensagem devolvida pela Camada de Núcleo do BioNimbuZ para uma dada ação. É verificado o código *HTTP* de resposta (*status code*), esperando-se, geralmente, o código `HTTP 200`, que indica sucesso na requisição. Outros códigos são comuns, tais como o código `HTTP 500`, sinalizando erro interno do servidor e `HTTP 400`, que significa que uma requisição foi mal montada (*Bad Request*);
- **Ações:** Definem o comando a ser executado pelo núcleo, enviando-lhe uma requisição, a fim de se obter uma resposta com os dados requeridos.

Assim, a partir dessas definições, foram desenvolvidas as implementações das `interfaces` `Action`, `ResponseInfo` e `RequestInfo`. A Tabela 5.7 apresenta a relação entre as classes `Action` x `ResponseInfo` x `RequestInfo`.

Tabela 5.7: Relação entre Classes Java e Páginas *HTML*.

Action	***ResponseInfo***	***RequestInfo***
`DeleteFile`	`DeleteFileResponse`	`DeleteFileRequest`
`GetWorkflowHistory`	`GetWorkflowHistoryResponse`	`GetWorkflowHistoryRequest`
`GetWorkflowStatus`	`GetWorkflowStatusResponse`	`GetWorkflowStatusRequest`
`Login`	`LoginResponse`	`LoginRequest`
`Logout`	`LogoutResponse`	`LogoutRequest`
`SignUp`	`SignUpResponse`	`SignUpRequest`
`StartWorkflow`	`StartWorkflowResponse`	`StartWorkflowRequest`
`Upload`	`UploadResponse`	`UploadRequest`

A Camada de Integração engloba a Camada de Aplicação e a Camada de Núcleo do BioNimbuZ. Dessa forma, foram necessárias alterações em ambas para que fosse possível haver comunicação entre esses dois sistemas. A Seção 5.5.1 trata das alterações realizadas na parte da Camada de Aplicação enquanto a Seção 5.5.2 apresenta o desenvolvimento necessário na Camada de Núcleo do BioNimbuZ, para que os dois sistemas fossem integrados.

5.5.1 Integração da Camada de Aplicação

Para permitir a execução de uma determinada *Action*, e seus respectivos *Requests* e *Responses*, foram desenvolvidas duas classes Java para controlar seu fluxo, executando uma

ação, enviando sua respectiva requisição e aguardando uma resposta. As responsabilidades dessas classes, chamadas `RestService` e `RestCommunicator`, estão listadas abaixo:

- **`RestService`:** Contém a lista de métodos disponíveis na Camada de Integração. Outros componentes de software só podem requisitar um comando *REST* a partir da instanciação desta classe, e caso o respectivo método tenha sido previamente implementado;
- **`RestCommunicator`:** É responsável por prover um passo a passo a ser seguido para realização da comunicação via *REST* da classe `RestService`. Utilizando o padrão de projeto *Strategy* [32], em conjunto com o conceito de polimorfismo, executa os seguintes métodos definido na interface *Action*:
  1. **`ping()`:** Verifica o estado da Camada de Núcleo do BioNimbuZ antes de toda requisição, efetuando o comando *ping* [63], podendo retornar verdadeiro (***`true`***) ou falso (***`falso`***). Em caso negativo, a requisição falha e retorna uma mensagem de erro ao usuário com a mensagem "*Erro Interno do Servidor*", sinalizando a falha na comunicação com o núcleo;
  2. **`setup()`:** Realiza a configuração prévia do objeto a ser enviado ao núcleo do BioNimbuZ, suprindo necessidades da comunicação via *REST*, tais como: instanciação do cliente *REST*, *set* das informações necessárias para a requisição, etc;
  3. **`prepareTarget()`:** Configura o endereço alvo (*target*) daquela requisição. Aponta para uma *URL* contendo o endereço IP do núcleo, juntamente com a porta que escuta requisições desse tipo e o caminho do recuro *REST* que irá recepcionar a requisição setada;
  4. **`execute()`:** Executa a ação desejada, realizando a operação *REST* definida pela requisição (*GET*, *POST*, *DELETE*, etc), retornando uma resposta para as classes de controle.

5.5.2 Integração da Camada de Núcleo do BioNimbuZ

Com a parte de envio das requisições desenvolvida na Camada de Aplicação, foi possível implementar a parte receptora dessas requisições, para que fossem processadas pela Camada de Núcleo do BioNimbuZ e devolvidas à aplicação.

Com esse objetivo, utilizou-se um *framework REST*, chamado *Resteasy* [65] para tratar as requisições vindas da Camada de Aplicação e enviar respostas para ela. A principal vantagem ao se desenvolver interfaces baseadas em *REST* é que sistemas com a capacidade de enviar requisições *HTTP* pela Internet podem ser integrados a outros com essa

capacidade. Dessa forma, a interface desenvolvida na Camada de Núcleo do BioNimbuZ é independente da aplicação que irá acessá-lo. Portanto, com o desenvolvimento dessa interface de comunicação, a Camada de Núcleo do BioNimbuZ pode receber requisições de qualquer sistema com tal capacidade, não apenas do sistema gerenciador de *workflows* desenvolvido neste trabalho.

Os elementos de software desenvolvidos para receberem as requisições foram chamados de *Resources* e cada um tem o objetivo de tratar uma entidade específica do sistema. Toda requisição é enviada a um determinado caminho (composto por endereço *IP*, porta e o caminho deste recurso *REST*), e cada *Resource* é configurado para receber as requisições deste caminho. Por exemplo, uma requisição enviada para o caminho `111.111.111.111:8080/rest/arquivos` vai ser recebida por aquele *Resource* configurado para receber dados do caminho `/rest/arquivos`.

Assim, foram projetados *Resources* para cada tipo de entidade do sistema, como usuários, arquivos e *workflows*. A Tabela 5.8 apresenta os *Resources* desenvolvidos, juntamente com os *Requests* que recepta e os *Responses* que envia.

Tabela 5.8: Recursos *REST* que Tratam as Requisições Vindas da Camada de Aplicação.

Resource	***Request* Tratada**	***Response* Enviada**
`UserResource`	`LoginRequest`	`LoginResponse`
	`LogoutRequest`	`LogoutResponse`
	`SignUpRequest`	`SignUpResponse`
`WorkflowResource`	`GetWorkflowHistoryRequest`	`GetWorkflowHistoryResponse`
	`GetWorkflowStatusRequest`	`GetWorkflowStatusResponse`
	`StartWorkflowRequest`	`StartWorkflowResponse`
`FileResource`	`DeleteFileRequest`	`DeleteFileResponse`
	`UploadRequest`	`UploadResponse`

Com esses *Resources* implementados, foi solucionado o problema de comunicação entre a Camada de Aplicação e a Camada de Núcleo do BioNimbuZ.

Os detalhes do desenvolvimento do método de acesso ao banco de dados utilizado pelo BioNimbuZ serão tratados na Seção 5.6.

5.6 Método de Acesso à Base de Dados do BioNimbuZ

Com o objeto de garantir a persistência e a rastreabilidade dos dados enviados ao BioNimbuZ, foi desenvolvido um método para acessar e manipular os dados gravados no banco

de dados *MySQL* [59], o qual foi escolhido por ser de simples instalação e configuração. Dessa forma, o usuário ao requisitar algum dado que a aplicação não possui, por exemplo o histórico de execução de determinado *workflow*, ela envia uma requisição pela camada de integração, pedindo ao núcleo do BioNimbuZ essa informação.

Para facilitar o desenvolvimento desse método de acesso a dados, foi utilizada a biblioteca de persistência *Hibernate* [64]. O *framework Hibernate* abstrai detalhes do acesso ao banco de dados, facilitando o desenvolvimento de aplicações que persistem seus dados. Nesse sentido, simplifica o desenvolvimento de operações de inserção, de recuperação, de atualização e de deleção (***CRUD*** - ***C****reate*, ***R****etrieve*, ***U****pdate*, ***D****elete*).

Para a construção dos métodos de manutenção da base de dados, foi utilizado o padrão de projeto *Data Access Object* [84]. Um Objeto de Acesso a Dados, usualmente chamado *DAO*, é um objeto que provê uma interface abstrata a algum tipo de banco de dados ou mecanismo de persistência, evitando duplicação de código e delegando a responsabilidade de acesso a base de dados à uma entidade específica da aplicação.

Assim, no contexto do BioNimbuZ, foram desenvolvidas três classe *DAO* responsáveis pelo acesso ao banco de dados correspondentes às entidades Usuário, Arquivos e *Workflows*. A Tabela 5.9 mostra a relação entre as classes *DAO* e as entidades que gerencia.

Tabela 5.9: Relação entre Classes *DAO* e Entidades Gerenciadas.

Classe *DAO*	**Entidade Gerenciada**
`UserDao`	`User`
`FileDao`	`FileInfo`
`WorkflowDao`	`Workflow`
	`Job`

Por fim, o fluxo de comunicação que ocorre quando a aplicação solicita um dado ao núcleo do BioNimbuZ é apresentado na Figura 5.4.

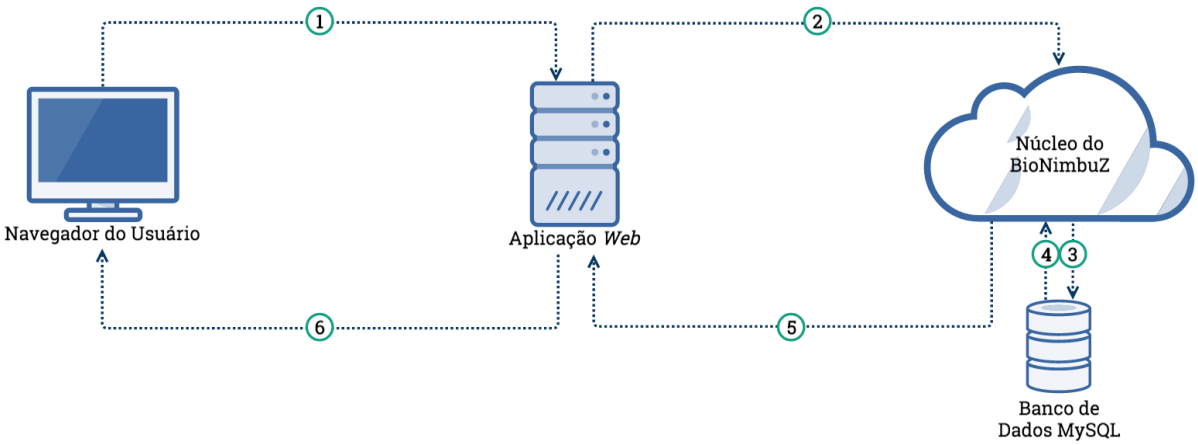

Figura 5.4: Fluxo de Comunicação para Operações no Banco de Dados.

Assim, a sucessão de operações que descrevem o acesso ao banco de dados, decorrente de uma soliticação do usuário, pode ser descrita da seguinte maneira:

1. Usuário requisita alguma informação que não está na Camada de Aplicação;
2. Aplicação envia uma requisição do tipo *Request* para a Camada de Núcleo do BioNimbuZ;
3. Núcleo do BioNimbuZ acessa o banco de dados;
4. Banco de dados retorna ao Núcleo a informação solicitada;
5. Núcleo envia uma mensagem de retorno do tipo *Response* para a Camada de Aplicação;
6. Aplicação formata os dados e envia ao usuário, por meio de uma página *HTML*.

5.7 Considerações Finais

Neste capítulo foram abordadas as etapas de desenvolvimento do sistema gerenciador de *workflows* científicos, proposto neste trabalho. Também foram apresentados os detalhes dos componentes de software que foram agregados à arquitetura do BioNimbuZ, sendo eles a Aplicação *Web*, a Camada de Integração e o Banco de Dados.

O próximo capítulo aborda os testes realizados com o objetivo de verificar o método de acesso desenvolvido, o funcionamento do sistema gerenciador a partir de *workflows* de Bioinformática, a persistência e a comunicação dos dados trafegados entre os componentes de software do BioNimbuZ.

Capítulo 6

Resultados Obtidos

Nesse capítulo serão mostrados os testes realizados para verificar o funcionamento do sistema gerenciador de *workflows* científicos desenvolvido e suas funcionalidades. Para isso, a Seção 6.1 descreve os principais objetivos dos testes realizados. A Seção 6.2 mostra as configurações das máquinas escolhidas para os testes e o *workflow* escolhido para tal. A Seção 6.3 descreve os procedimentos realizados com o objetivo de testar a solução desenvolvida. A Seção 6.4 apresenta o sistema gerenciador de *workflows* científicos desenvolvido neste trabalho e os resultados obtidos com o mesmo, a partir da execução do *workflow* de testes proposto. Nessa seção, também são apresentadas a interface percebida pelo usuário para a criação do *workflow*, e os testes escolhidos. Por último, a Seção 6.5 apresenta as considerações finais, realizadas com base nos resultados obtidos.

6.1 Objetivos dos testes

Foi realizado um teste com um *workflow* real de Bioinformática, o qual é descrito na próxima seção, verificando se o objetivo de melhoria no gerenciamento de *workflows* foi alcançado pelo sistema proposto. Verificou-se também a usabilidade do sistema quanto às funcionalidades desenvolvidas, tais como: *upload* de arquivos; *design* gráfico de *workflows* e gerenciamento dos *workflows* submetidos pelo usuário.

Conforme foi observado durante a execução, a camada de integração não apresentou falhas, dado que as requisições e respostas estavam sendo enviadas e recebidas, respectivamente, pelos componentes envolvidos. Quanto à persistência das informações, também objetivo do presente trabalho, verificou-se junto ao banco de dados se a mesma foi cumprida, a partir de consultas (*queries*) eventuais à base de dados.

Os testes realizados tinham quatro objetivos, sendo eles:

- Analisar o ganho de produtividade e facilidade de utilização da plataforma BioNimbuZ, possibilitando que os usuários gerenciassem seus *workflows* de maneira simples, além de projetá-los graficamente;
- Analisar o comportamento do sistema gerenciador com a submissão de um *workflow* real de bioinformática;
- Garantir que a troca de mensagens entre a aplicação *web* e o núcleo do BioNimbuZ estava ocorrendo corretamente, a partir da camada de integracão proposta;
- Garantir que os dados enviados e gerados pela plataforma estavam sendo persistidos, por meio da utilização do banco de dados MySQL.

6.2 Ambiente de Testes e *Workflow* Escolhido

Nessa seção, são apresentados os passos envolvidos para analisar o comportamento do sistema gerenciador de *workflows* desenvolvido neste trabalho, com a execução de um *workflow* real de Bioinformática. Para isso, a Seção 6.2.1 descreve a infraestrutura projetada com tal objetivo e a Seção 6.2.2 retrata os detalhes do *workflow* escolhido.

6.2.1 Configuração do Ambiente

Para testar o sistema gerenciador de *workflow* proposto neste trabalho e o comportamento da federação de nuvens, foram utilizadas duas máquinas da Universidade de Brasília e quatro na infraestrutura de nuvem computacional da *Amazon*, a *Amazon Elastic Compute Cloud* (EC2) [39]. A Tabela 6.1 apresenta as configurações presentes nas máquinas da Universidade de Brasília e também as configurações das instâncias que foram executadas na *Amazon* EC2.

Tabela 6.1: Configuração dos computadores utilizados para execução dos testes.

Infraestrutura	**Item de Configuração**	**Configuração**
Universidade de Brasília	Processador	Intel Core i7 3.1 GHz
	Memória	8 *Gigabytes* de Memória RAM
	Armazenamento	1 Terabyte
	Sistema Operacional	Linux Ubuntu 14.04
Amazon EC2	Processador	Intel Xeon E5-2680 v3 2.6 GHz
	Memória	3.75 *Gigabytes* de Memória RAM
	Armazenamento	1 Terabyte
	Sistema Operacional	Linux Ubuntu 14.04

6.2.2 *Workflow* e Dados Utilizados

Com o objetivo de testar a solução desenvolvida, foram executados testes com um *workflow* real de Bioinformática, que tem o objetivo de identificar genes diferencialmente expressos em células humanas cancerosas do rim e do fígado [51] [67], com fragmentos gerados pelo sequenciador Illumina [38]. O *workflow* consiste em quatro etapas, as quais são apresentadas na Figura 6.1.

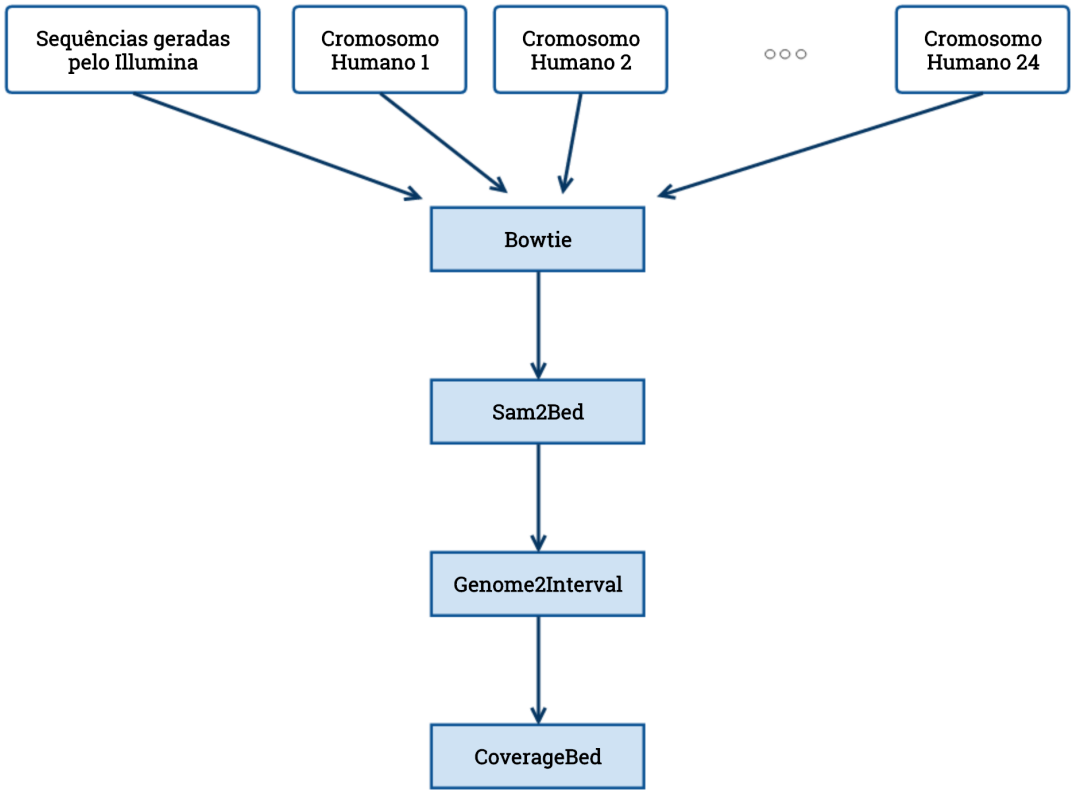

Figura 6.1: *Workflow* Utilizado nos Testes.

A ferramenta **Bowtie** [80] é utilizada para mapear os fragmentos nos 24 cromossomos humanos de referência, isto é, o software identifica a região do genoma de referência de onde cada fragmento de entrada está localizado. Assim, quando um conjunto de fragmentos é mapeado na mesma região, pode-se inferir que possuem a mesma organização estutural do genoma de referência.

Na segunda fase é realizada uma conversão de tipos de dados, na qual o arquivo de saída do Bowtie (arquivo .SAM) é convertido para o formato .BED. Para isso, é utilizado um *script* de conversão, chamado ***sam2bed*** [56].

Logo após, na terceira etapa, é utilizado um *script* desenvolvido por Saldanha [3] chamado ***genome2interval*** para gerar intervalos de tamanho fixo baseados no tamanho de cada cromossomo, os quais serão utilizados na fase seguinte.

Por fim, no quarto passo, são gerados histogramas que indicam o número de fragmentos mapeados por intervalo dos cromossomos com a ferramenta ***coverageBed***, da *suite* BEDTools [62].

Os 24 cromossomos do genoma de referência humano (**hg19**) foram obtidos do banco de dados NCBI (*National Center for Biotechnology Information*) [55] e a descrição do genoma pode ser obtida em `http://www.ncbi.nlm.nih.gov/genome/assembly/293148/`.

6.3 Descrição dos Testes Realizados

Os testes realizados tinham como objetivo testar as funcionalidades disponibilizadas na aplicação desenvolvida, tais como envio de arquivo, visualização do armazenamento disponível para uso, criação e gerenciamento dos *workflows*, etc.

Primeiramente, foram iniciadas as duas máquinas físicas do ambiente de execução da Universidade de Brasília, logo após foram iniciadas as quatro máquinas virtuais da infraestrutura da Amazon EC2. Com o ambiente de testes em execução, foram enviados os arquivos executáveis (*.JAR*) do BioNimbuZ às máquinas e postos em execução.

Em uma das máquinas da Amazon EC2 foi iniciado o banco de dados *MySQL* e, para que a persistência fosse realizada de maneira correta, as outras instâncias do BioNimbuZ apontavam para essa máquina como servidor de banco de dados. Dessa forma, a persistência ocorria centralizadamente em uma das instâncias (esse ponto deve ser abordado e otimizado em trabalhos futuros, os quais serão sugeridos no Capítulo 7). A estrutura de tabelas do banco de dados é criada pelo *framework Hibernate* no momento de sua execução, a partir do mapeamento feito nas classes de modelo (detalhadas no Capítulo 5).

Em uma das máquinas da Universidade de Brasília, a aplicação *web* foi posta em execução. Assim, cinco máquinas executavam o núcleo do BioNimbuZ com foco em poder computacional, sendo que uma delas executava o banco de dados, e a outra executava o sistema gerenciador de *workflows*.

Assim, com as máquinas, o banco de dados e a aplicação *web* em execução, o teste seguiu o seguinte procedimento:

1. Primeiramente, foi realizado o *login* na plataforma;
2. Em seguida, foi realizado o *upload* do arquivo de entrada do primeiro passo do *workflow* que era o resultado do sequenciamento do Illumina. O arquivo de referência era o cromossomo humano, o qual já estava armazenado no BioNimbuZ;
3. Após o envio do arquivo, foi iniciado o projeto e *design* do *workflow* descrito neste capítulo. A partir da ligação do elementos, foram definidos os parâmetros de entrada

(arquivos de entrada, parâmetros de execução, arquivos de referência e formato do arquivo de saída);

4. Após o *design* do *workflow*, o mesmo foi submetido ao BioNimbuZ para ser executado;

5. Após o envio à plataforma, foi verificado o *status* de sua execução, de seu início até sua correta finalização.

Na próxima Seção serão mostradas as telas desenvolvidas, seguindo o passo a passo descrito acima para conclusão dos testes.

6.4 Sistema Gerenciador de *Workflows* Científicos

Nesta seção, serão apresentadas as telas desenvolvidas, as quais servem de interface para os serviços disponíveis no BioNimbuZ. Será apresentado como o *workflow* usado nos testes foi implementado no ambiente de testes e a maneira na qual suas saídas são apresentadas ao usuário.

6.4.1 *Login*

Primeiramente, para acessar o sistema, é necessário realizar o *login* no mesmo. A Figura 6.2 mostra a tela inicial da aplicação *web* desenvolvida. Nessa tela, o usuário deve entrar com seu usuário e senha cadastrados previamente para ter acesso aos recursos disponíveis no BioNimbuZ.

Figura 6.2: Tela de *Login* da Aplicação *Web*.

6.4.2 Tela Inicial

Ao realizarem o *login*, os usuários que possuírem acesso ao ambiente *web* visualizam a tela mostrada na Figura 6.3. Nessa tela, e em todas as outras, é mostrado um *menu* com as seguintes funcionalidades disponíveis ao usuário:

- ***Workflows*:** Funcionalidades referentes ao gerenciamento dos *workflows* do usuário, tais como:
  - **Criar novo *Workflow*:** A partir dessa opção, o usuário pode iniciar ou importar um *workflow*;
  - ***Status* de seus *Workflows*:** Possibilita ao usuário visualizar o estado e o histórico de execução de seus *workflows*.
- **Armazenamento:** Funcionalidades referentes ao gerenciamento do armazenamento do usuário, sendo elas:
  - **Meu Armazenamento:** Nessa opção, o usuário pode verificar a utilização de sua cota de armazenamento (definida como 1 *Gigabyte*);
  - **Enviar Arquivo:** Possibilita ao usuário realizar o *upload* de arquivos à plataforma;

– **Deletar Arquivo:** Opção referente à deleção de arquivos do usuário;

– **Realizar *Download*:** Utilizada caso o usuário queira fazer o *download* dos arquivos enviados à plataforma ou dos arquivos gerados como saída de seus *workflows*.

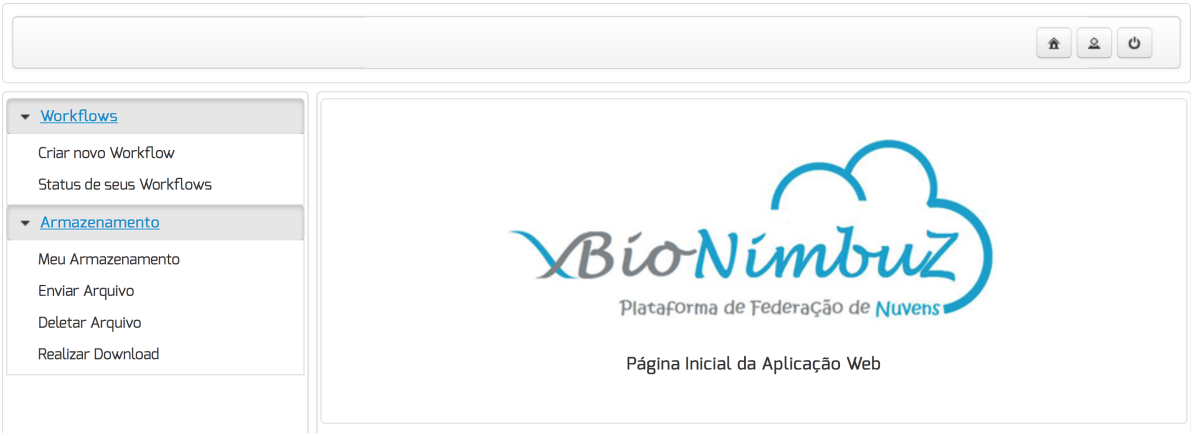

Figura 6.3: Tela Inicial da Aplicação *Web*.

6.4.3 Início do Projeto dos *Workflows*

Na tela mostrada na Figura 6.4, o usuário inicia o projeto de seu *workflow*, podendo escolher entre criar um novo *workflow* ou importar um anteriormente criado. A aplicação *web* possibilita, ao fim do projeto de um *workflow*, exportá-lo. Assim, com esse *workflow* exportado, é possível compartilhá-lo com outros usuários do sistema.

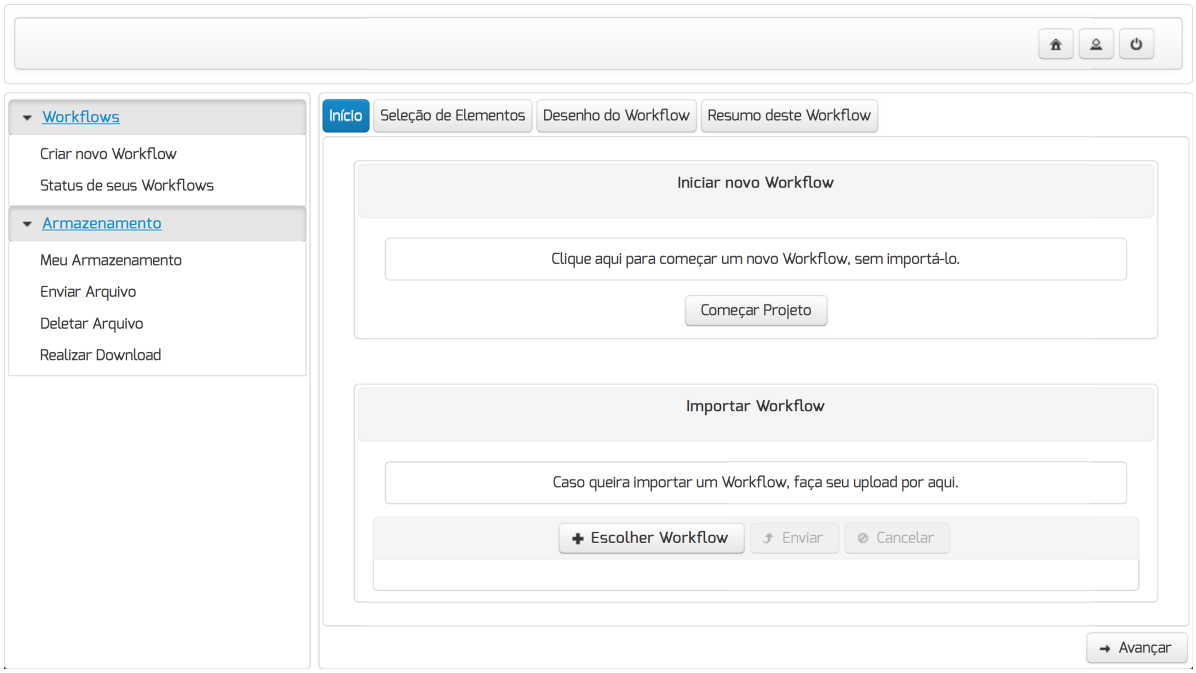

Figura 6.4: Tela inicial de criação dos *workflows*.

6.4.4 Seleção de Elementos

Ao clicar em "**Começar Projeto**", o usuário então é direcionado para a página de seleção de elementos, mostrada na Figura 6.5. Nela, o mesmo poderá planejar a execução de seu *workflow*, escolhendo quais elementos irão compor este fluxo. A lista de elementos disponíveis é mantida no núcleo do BioNimbuZ e é enviada à aplicação quando esta é iniciada. À medida que o usuário escolhe os elementos, eles são adicionados na listagem "**Elementos Escolhidos**", podendo o mesmo excluí-los, caso não os queira mais. Para o *workflow* testado nesse capítulo, foram escolhidos os quatro passos necessários: Bowtie, Sam2Bed, Genome2Interval e CoverageBed.

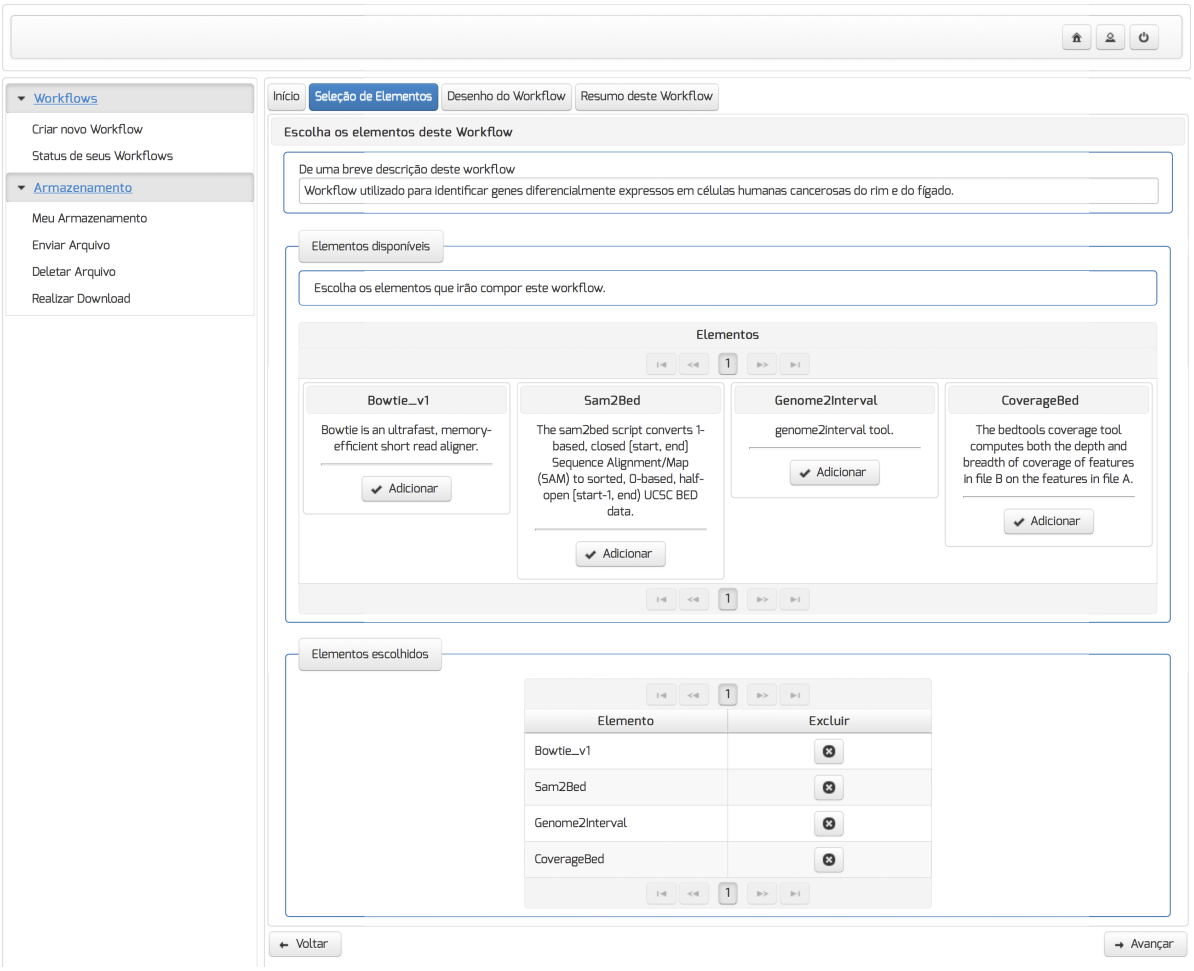

Figura 6.5: Tela Utilizada para Selecionar Elementos que Irão Compor o *Workflow*.

6.4.5 *Design* do *Workflow*

Com os elementos definidos na tela anterior (Figura 6.5), o usuário é redirecionado para a tela seguinte (mostrada na Figura 6.6), passando para a fase de *design* do *workflow*. Nela, são apresentados os elementos escolhidos pelo usuário dispostos de maneira gráfica, conforme mostra a Figura 6.6. Para o *workflow* definido, foram feitas as ligações conforme necessário para sua correta execução.

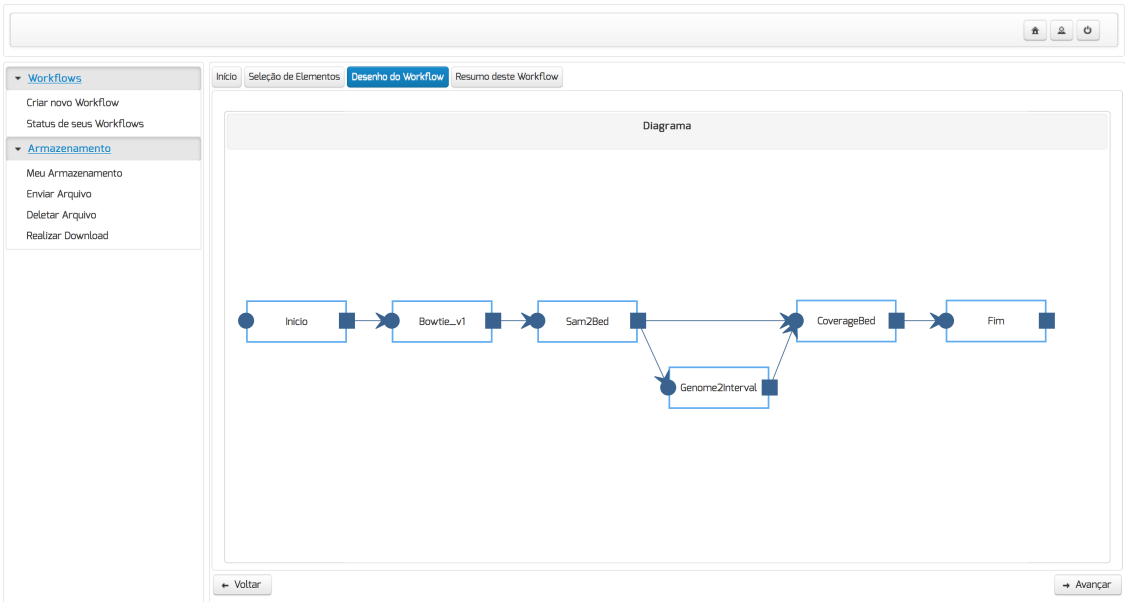

Figura 6.6: Passo de *Design* do Fluxo do *Workflow*.

Com isso, o usuário pode satisfazer as dependências de cada elemento, arrastando os pontos de ligação, definir suas entradas (arquivos enviados), definir os parâmetros de execução (argumentos daquele componente), escolher algum arquivo de referência (o qual já se encontra na plataforma), ou apenas sinalizar que a entrada de um determinado elemento será a saída de outro. Com esse objetivo, ao se conectar dois elementos, é apresentada uma caixa de diálogo (Figura 6.7) pedindo estes dados.

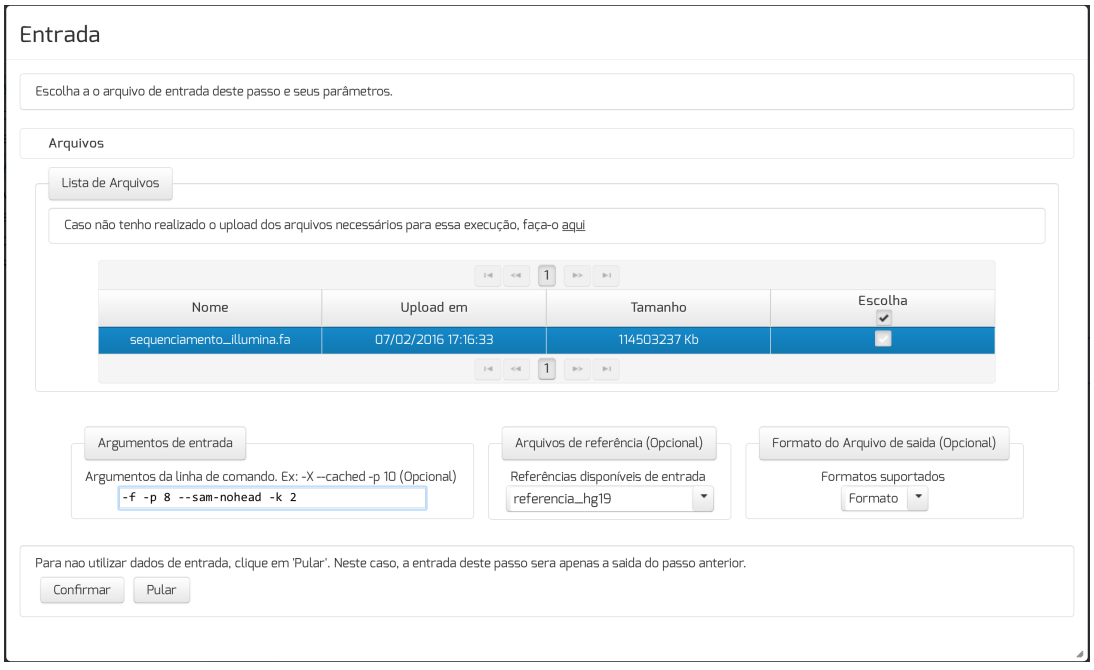

Figura 6.7: Tela para Definição de Parâmetros de Execução de um Elemento do *Workflow*.

6.4.6 Confirmação

Com todos os elementos escolhidos e com suas dependências (entrada, saída e parâmetros de execução) satisfeitas, é apresentada uma tela de confirmação (veja a Figura 6.8), em que o usuário pode confirmar os dados daquele *workflow* e submetê-lo à execução, ou pode apenas realizar o *download* do arquivo ***.flow*** (representação do objeto Java contendo os dados de um *workflow*). De posse deste arquivo, o usuário pode importá-lo em outro momento, ou pode também compartilhá-lo com outros usuários do sistema, para que outros tenham acesso ao seu trabalho.

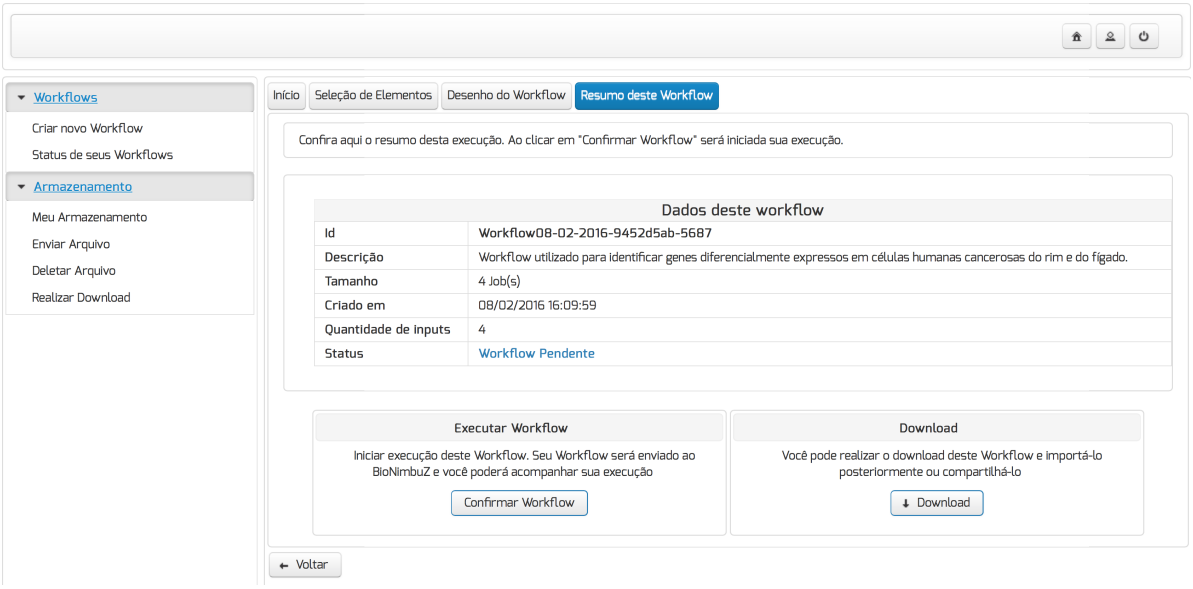

Figura 6.8: Tela Final da Composição do *Workflow*.

6.4.7 *Status* dos *Workflows* Submetidos

A partir do momento em que o usuário escolheu submeter o *workflow* à plataforma BioNimbuZ, ele pode visualizar o estado de sua execução na tela de *status*, apresentada na Figura 6.9. Os possíveis estados de um *workflow* estão retratados na Tabela 6.2.

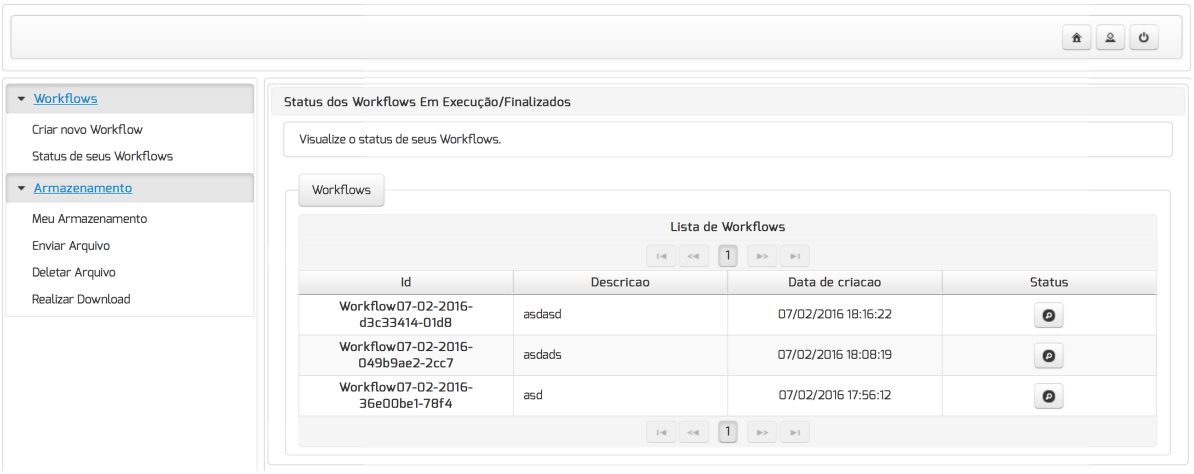

Figura 6.9: Tela final da composição do *workflow*.

Tabela 6.2: Estados Possíveis de um *Workflow*

Status	**Nível**	**Descrição**
Workflow Pausado	`Informativo`	Foi requisitada a pausa na execução do *Workflow*.
Workflow Pendente	`Informativo`	Seu *Workflow* está agurdando execução pelo BioNimbuZ.
Workflow em Execução	`Informativo`	O *Workflow* está em plena execução.
Workflow Finalizado com Sucesso	`Informativo`	Seu *Workflow* foi finalizado sem alertas e sem erros.
Workflow Finalizado com Alertas	`Alerta`	Alertas foram lançados durante a execução deste *Workflow*.
Workflow Finalizado com Erros	`Erro`	Erros nao permitiram que este *Workflow* terminasse com sucesso.
Workflow Parado com Erros	`Erro`	Estado parcial com erros.

6.4.8 Histórico de Execução dos *Workflows*

Para que fosse possível ao usuário rastrear a execução de determinado *workflow* e realizar o *download* dos arquivos de saída gerados por ela, foi necessária a inserção de dados no banco de dados a cada passo executado pelo sistema (*log*). Para isso, a tela apresentada na Figura 6.10 foi desenvolvida, contendo esses *logs* e também os arquivos de saída disponíveis para *download*. Ao clicar em "***Download***", o navegador do usuário inicia a transferência do arquivo do BioNimbuZ para a máquina local do usuário.

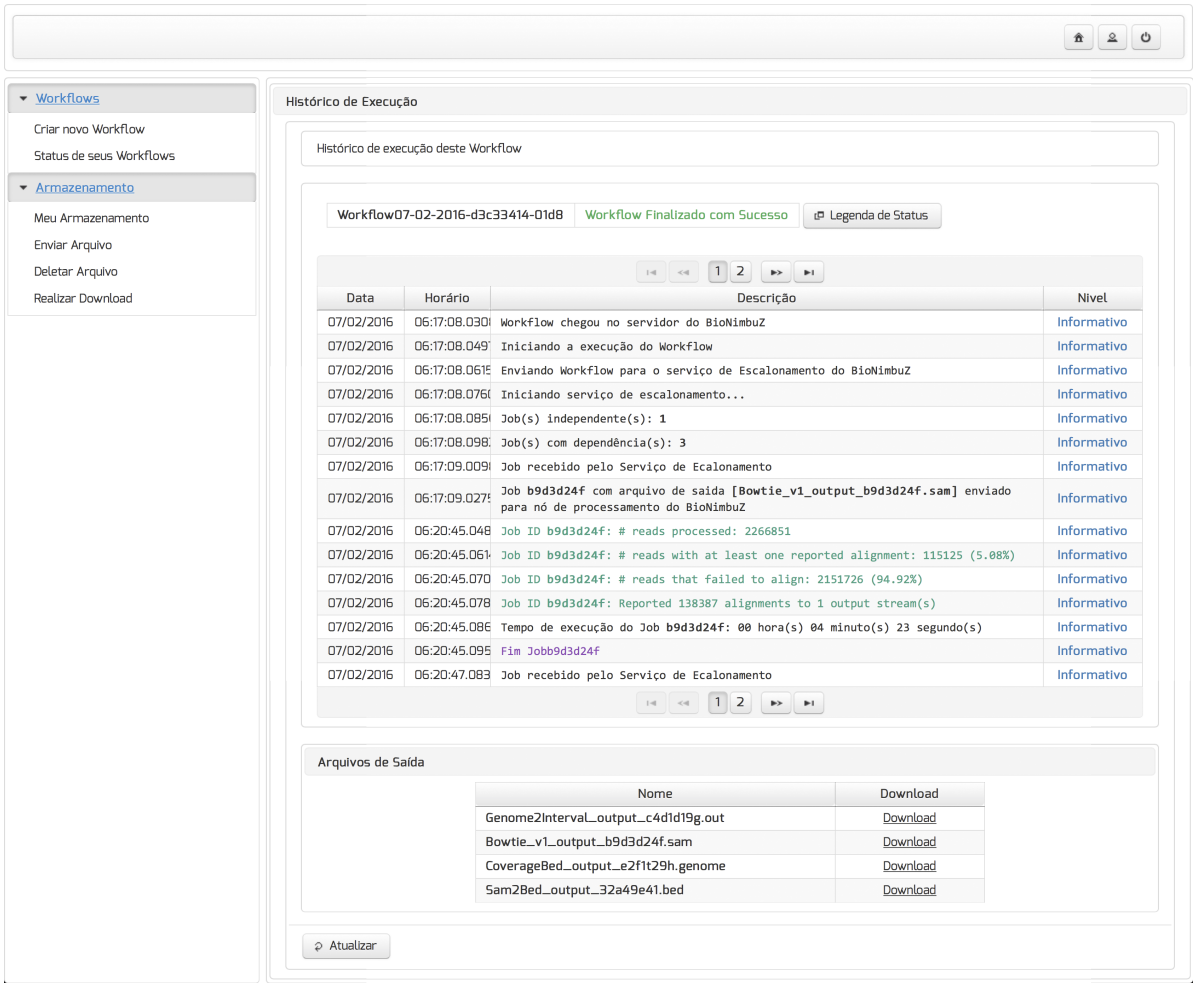

Figura 6.10: Tela com as Informações da Execução de *Workflow*.

6.4.9 *Upload* e Controle do Armazenamento dos Arquivos Enviados

Com o objetivo de compor um *workflow*, o usuário tem de prover dados de entrada. Para isso, deve enviar seus arquivos para a plataforma do BioNimbuZ, a partir da tela mostrada na Figura 6.11.

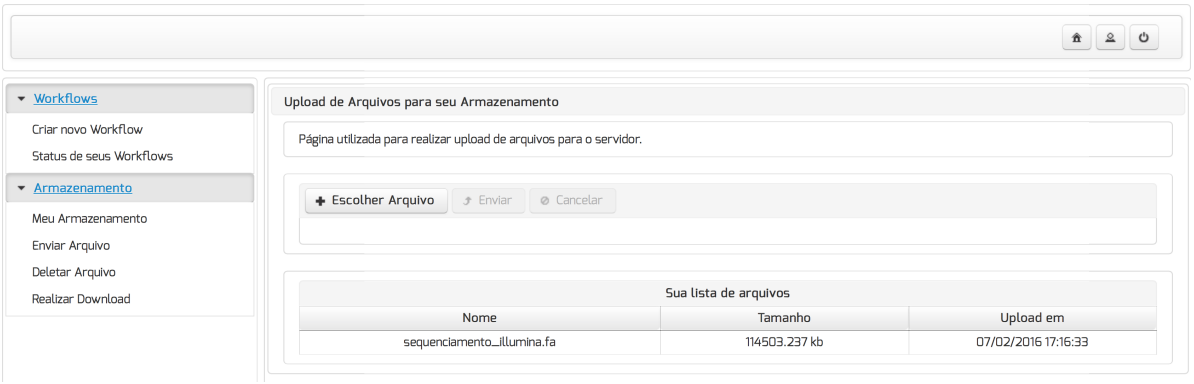

Figura 6.11: Tela Utilizada para Realizar o *Upload* dos Arquivos do Usuário.

Nesta implementação foi definido um armazenamento máximo de 1024 *Megabytes* por usuário (1GB). Assim, para ter controle de seu armazenamento, foi desenvolvida a interface apresentada na Figura 6.12.

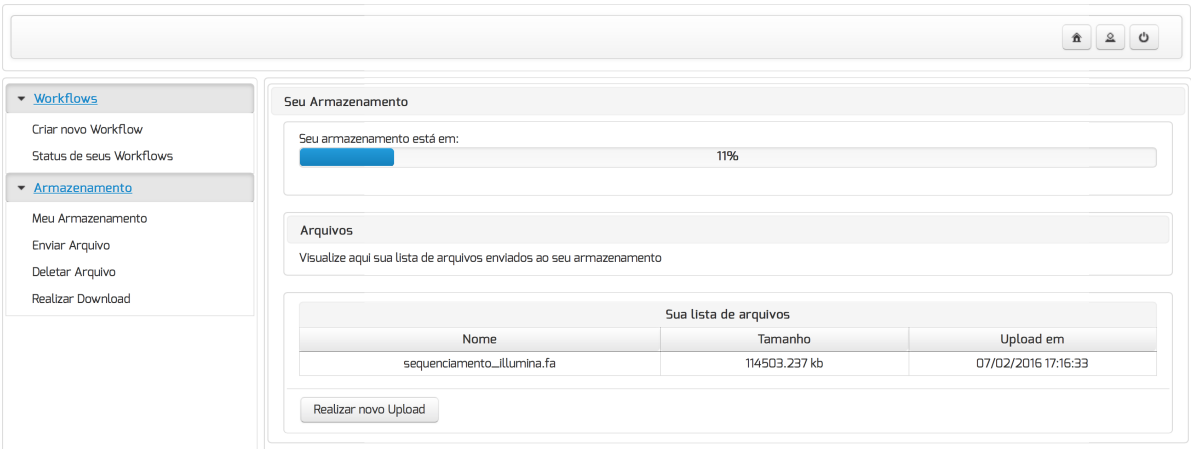

Figura 6.12: Tela na qual os Usuários Controlam seu Armazenamento e Verificam sua Porcentagem de Utilização.

6.5 Considerações Finais

Neste capítulo foram apresentados os testes realizados na plataforma BioNimbuZ e o gerenciamento de *workflows* pelo usuário, a partir do sistema gerenciador de *workflows* científicos, o qual foi principal intuito deste trabalho.

Dessa forma, foi verificado que os objetivos delineados no presente capítulo foram cumpridos nos testes, tendo em vista os resultados obtidos, e a melhoria no gerenciamento de *workflows* submetidos pelos usuários do BioNimbuZ.

Capítulo 7

Conclusões e Trabalhos Futuros

Neste trabalho foi proposto um sistema gerenciador de *workflows* científicos para a plataforma de federação de nuvens BioNimbuZ, de modo a ampliar as funcionalidades do sistema e aprimorar o gerenciamento dos *workflows* submetidos a ele. A finalidade da solução desenvolvida foi também integrar a aplicação *web* desenvolvida ao núcleo do BioNimbuZ e a persistência das informações trafegadas na plataforma.

Foi proposta uma nova arquitetura, a qual engloba o núcleo do BioNimbuZ e dois novos componentes, sendo eles a aplicação *web* e a camada de integração baseada em *webservices REST*. Com a aplicação *web*, foi verificada uma melhoria no método de acesso aos recursos do BioNimbuZ, antes através de comandos executados em um *terminal*, facilitando sua utilização, tornando-o acessível por qualquer usuário conectado à Internet e provendo meios para que fosse possível o gerenciamento de *workflows*.

Foram realizados testes com o sistema gerenciador de *workflows*, realizando-se o projeto, a composição e a execução de um *workflow* com objetivo de identificar genes diferencialmente expressos em células humanas cancerosas do rim e do fígado. Também foram realizados testes que mostraram que a aplicação *web* realiza o controle de acesso de usuários ao sistema, verificando suas credenciais no momento de *login*.

Como trabalhos futuros, sugere-se a utilização de bancos de dados distribuídos, como por exemplo o Cassandra [30], a fim de tornar os dados persistidos na plataforma mais escalável e altamente disponíveis. A utilização de bancos de dados distribuídos também visa aumentar a tolerância à falhas do sistema, visto que a queda de alguns nós não indisponibilizaria os serviços providos pelo BioNimbuZ.

Os arquivos de referência de Bioinformática são geralmente muito grandes, chegando a 20 Gigabytes cada. Dessa forma, propõe-se a integração de serviços de armazenamento na nuvem, como o Amazon S3 [72], com o objetivo de manter as máquinas do BioNimbuZ focados em poder computacional para melhor executar os *workflows* submetidos, delegando a responsabilidade do armazenamento a outro componente de software.

Por fim, tendo em vista a nova abordagem de gerenciamento de *workflows* baseada na aplicação desenvolvida, tornou-se necessário o desenvolvimento de novas camadas de segurança, prevendo e evitando possíveis ataques à plataforma do BioNimbuZ.

Referências

[1] I. Altintas, O. Barney, e E. Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. *International Provenance and Annotation Workship (IPAW'06)*, 2006. 19

[2] Amazon. Amazon. `http://www.amazon.com`. Acessado online em 19 de janeiro de 2016. 11

[3] Özalp Babaoğlu e Keith Marzullo. *Distributed Systems (2Nd Ed.)*, chapter Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms, pages 55–96. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1993. 5

[4] C. A. L. Borges. Escalonamento de tarefas em uma infraestrutura de computação em nuvem federada para aplicações em bioinformática, 2011. Monografia apresentada para conclusão do Curso de Computação. 41

[5] Christopher Brooks, Edward A. Lee, Xiaojun Liu, Stephen Neuendorffer, Yang Zhao, e Haiyang Zheng. Heterogeneous concurrent modeling and design in java (volume 2: Ptolemy ii software architecture). Technical Report UCB/EECS-2008-29, EECS Department, University of California, Berkeley, Apr 2008. 26

[6] Rajkumar Buyya, Christian Vecchiola, e S. Thamarai Selvi. *Mastering Cloud Computing: Foundations and Applications Programming*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013. 1

[7] Antonio Celesti, Francesco Tusa, Massimo Villari, e Antonio Puliafito. How to enhance cloud architectures to enable cross-federation. In *Proceedings of the 2010 IEEE 3rd International Conference on Cloud Computing*, CLOUD '10, pages 337–345, Washington, DC, USA, 2010. IEEE Computer Society. 1, 16, 17

[8] Shirley Cohen, Sarah Cohen-Boulakia, e Susan Davidson. Towards a Model of Provenance and User Views in Scientific Workflows. *Lecture Notes in Computer Science*, 4075:264–279, 2006. 19

[9] CollabNet. Cloudforge. `http://www.cloudforge.com`. Acessado online em 19 de janeiro de 2016. 15

[10] George Coulouris, Jean Dollimore, Tim Kindberg, e Gordon Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition, 2011. 5

[11] V. Curcin e M. Ghanem. Scientific workflow systems - can one size fit all? In *Biomedical Engineering Conference, 2008. CIBEC 2008. Cairo International*, pages 1–9, 2008. xi, 22, 23, 24, 25, 26, 28

[12] Datapipe. Gogrid. https://www.datapipe.com/gogrid/. Acessado online em 19 de janeiro de 2016. 14

[13] B. R. de Moura e D. L. Bacelar. Política para armazenamento de arquivos no zoonimbus, 2013. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 24, 32, 33, 41

[14] W. de O. B. Júnior. Escalonador de tarefas para o plataforma de nuvens federadas bionimbuz usando beam search iterativo multiobjetivo, 2016. Monografia apresentada para conclusão do Curso de Engenharia da Computação. 41

[15] G. S. S. de Oliveira. Acosched: um escalonador para o ambiente de nuvem federada zoonimbus, 2013. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 32, 33, 41

[16] Ewa Deelman, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Sonal Patil, Mei-Hui Su, Karan Vahi, e Miron Livny. *Grid Computing: Second European AcrossGrids Conference, AxGrids 2004, Nicosia, Cyprus, January 28-30, 2004. Revised Papers*, chapter Pegasus: Mapping Scientific Workflows onto the Grid, pages 11–20. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. 23, 33, 34

[17] Read The Docs. Galaxy project interface. http://mgescan.readthedocs.org/en/. Acessado online em 15 de fevereiro de 2016. xi, 29

[18] Dropbox. Dropbox. http://www.dropbox.com/. Acessado online em 12 de janeiro de 2016. 16

[19] Research Studio Digital Memory Engineering. Research studio digital memory engineering. http://www.rs-dme.at/download/triana-screenshot1.png. Acessado online em 29 de janeiro de 2016. xi, 26

[20] Roy T. Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, University of California, Irvine, Irvine - Irvine, CA 92697, USA, 2000. 56

[21] Roy T. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, e T. Berners-Lee. Rfc 2616, hypertext transfer protocol – http/1.1, 1999. 33, 53

[22] Center for Comparative Genomics e Bioinformatics at Penn State University. Galaxy project. https://galaxyproject.org. Acessado online em 15 de fevereiro de 2016. 28

[23] Ian Foster, Carl Kesselman, e Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *IJHPCA*, 15(3):200–222, 2001. 8

[24] Ian Foster, Yong Zhao, Ioan Raicu, e Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10. IEEE, November 2008. xi, 5, 8, 9, 10, 12, 14

[25] Linux Foundation. Kvm. http://www.linux-kvm.org/page/Main_Page. Acessado online em 11 de janeiro de 2016. 13

[26] Linux Foundation. Xen project. http://www.xenproject.org. Acessado online em 11 de janeiro de 2016. 10, 13

[27] The Apache Software Foundation. Apache foundation. http://apache.org/. Acessado online em 15 de janeiro de 2016. 33

[28] The Apache Software Foundation. Apache thrift. https://thrift.apache.org. Acessado online em 10 de janeiro de 2016. 34

[29] The Apache Software Foundation. Apache zookeeper. http://zookeeper.apache.org/. Acessado online em 15 de janeiro de 2016. xi, 33, 34, 35, 36

[30] The Apache Software Foundation. Cassandra. http://cassandra.apache.org. Acessado online em 02 de fevereiro de 2016. 76

[31] The Apache Software Foundation. Struts. https://struts.apache.org. Acessado online em 18 de janeiro de 2016. 51

[32] Elisabeth Freeman e Eric Freeman. *Use a cabeça! Padrões de Projeto*. Alta Books, Brasil, 2007. xi, 33, 50, 58

[33] E. Gallopoulos, E. Houstis, e J.R. Rice. Computer as Thinker/Doer: Problem-Solving Environments for Computational Science. *Computational Science Engineering, IEEE*, 1(2):11–23, Summer 1994. 25

[34] Moustafa Ghanem, Vasa Curcin, Patrick Wendel, e Yike Guo. *Building and Using Analytical Workflows in Discovery Net*, pages 119–139. John Wiley & Sons, Ltd, 2009. 23

[35] Seyyed Mohsen Hashemi e Amid Khatibi Bardsiri. Cloud computing vs. grid computing. *ARPN Journal of Systems and Softwares*, 2(5):188–194, May 2012. 8

[36] B. Hoehrmann. Scripting Media Types. RFC 4329 (Informational), April 2006. 53

[37] D Hull, K Wolstencroft, R Stevens, C Goble, M R Pocock, P Li, e T Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Res*, 34(Web Server issue):729–732, July 2006. 23

[38] Inc Illumina. Illumina. http://www.illumina.com. Acessado online em 04 de fevereiro de 2016. 64

[39] Amazon Inc. Amazon ec2. https://aws.amazon.com/pt/ec2/. Acessado online em 12 de janeiro de 2016. 14, 16, 30, 63

[40] AppScale Systems Inc. Appscale. http://www.appscale.com. Acessado online em 19 de janeiro de 2016. 15

[41] Elsevier Inc. Interface gráfica do sistema kepler. http://www.cell.com/cms/attachment/2002983292/2011324825/gr2.jpg. Acessado online em 30 de janeiro de 2016. xi, 28

[42] Google Inc. Google. http://www.google.com.br. Acessado online em 19 de janeiro de 2016. 11

[43] Google Inc. Google compute engine. https://cloud.google.com/compute/. Acessado online em 19 de janeiro de 2016. 14

[44] Google Inc. Protocol buffers. https://developers.google.com/protocol-buffers/. Acessado online em 21 de janeiro de 2016. 34

[45] H. Lie, B. Bos, e C. Lilley. The text/css Media Type. RFC 2318 (Informational), March 1998. 53

[46] Abiquo Europe Ltd. Abiquo. http://www.abiquo.com. Acessado online em 19 de janeiro de 2016. 15

[47] ID Business Solutions Ltd. Idbs. https://www.idbs.com/en/. Acessado online em 19 de janeiro de 2016. 23

[48] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, e Yang Zhao. Scientific workflow management and the kepler system: Research articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, August 2006. xi, 19, 23, 26, 27, 28

[49] Bertram Ludäscher, Ilkay Altintas, Shawn Bowers, Julian Cummings, Terence Critchlow, Ewa Deelman, David D. Roure, Juliana Freire, Carole Goble, Matthew Jones, Scott Klasky, Timothy McPhillips, Norbert Podhorszki, Claudio Silva, Ian Taylor, e Mladen Vouk. *Scientific Process Automation and Workflow Management*, chapter 13. Computational Science Series. Chapman & Hall, Boca Raton, 2009. 2, 19, 21

[50] S. Majithia, M. Shields, I. Taylor, e I. Wang. Triana: a graphical web service composition and execution toolkit. In *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 514–521, July 2004. 25

[51] John C. Marioni, Christopher E. Mason, Shrikant M. Mane, Matthew Stephens, e Yoav Gilad. RNA-Seq: An assessment of technical reproducibility and comparison with gene expression arrays. *Genome Research*, 18:1509–1517, 2008. 64

[52] Peter Mell e Timothy Grance. The nist definition of cloud computing. Technical Report 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011. 7, 11, 12

[53] Microsoft. Microsoft. http://www.microsoft.com. Acessado online em 19 de janeiro de 2016. 11

[54] R. Morrison. Cluster computing: Architectures, operating systems, parallel processing & programming languages, 2003. xi, 6, 7

[55] National center for Biotechnology Information. NCBI. http://www.ncbi.nlm.nih.gov. Acessado online em 06 de fevereiro de 2016. 65

[56] Shane Neph. Sam2Bed. http://bowtie-bio.sourceforge.net/index.shtml. Acessado online em 05 de fevereiro de 2016. 64

[57] OnlineHPC. Taverna online. http://onlinehpc.com/. Acessado online em 30 de janeiro de 2016. xi, 25

[58] Oracle. Java server faces. https://javaserverfaces.java.net/. Acessado online em 12 de janeiro de 2016. 51

[59] Oracle. Mysql. www.mysql.com. Acessado online em 02 de fevereiro de 2016. 51, 60

[60] Play. Play framework. https://www.playframework.com. Acessado online em 18 de janeiro de 2016. 51

[61] PrimeTek. Primefaces. http://www.primefaces.org/. Acessado online em 12 de janeiro de 2016. 53

[62] Aaron R. Quinlan. BedTools. http://bedtools.readthedocs.org/en/latest/. Acessado online em 06 de fevereiro de 2016. 65

[63] J. Quittek e K. White. Definitions of Managed Objects for Remote Ping, Traceroute, and Lookup Operations. RFC 4560 (Proposed Standard), June 2006. Acessado online em 01 de fevereiro de 2016. 58

[64] JBoss RedHat. Hibernate. http://hibernate.org/. Acessado online em 02 de fevereiro de 2016. 60

[65] JBoss RedHat. Resteasy. http://resteasy.jboss.org. Acessado online em 02 de fevereiro de 2016. 58

[66] Bhaskar Prasad Rimal, Eunmi Choi, e Ian Lumb. A taxonomy and survey of cloud computing systems. In *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, NCM '09, pages 44–51, Washington, DC, USA, 2009. IEEE Computer Society. 1

[67] Mark D Robinson e Alicia Oshlack. A scaling normalization method for differential expression analysis of RNA-Seq data. *Genome Biol*, 11(3):R25, 2010. 64

[68] H. V. Saldanha. Bionimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de bioinformática. Master's thesis, Departamento de Ciência de Computação, Universidade de Brasília, 2012. 2, 32, 33, 44, 47

[69] Salesforce.com. Heroku. https://www.heroku.com/. Acessado online em 12 de janeiro de 2016. 15, 16

[70] Salesforce.com. Salesforce. http://www.salesforce.com/br/. Acessado online em 19 de janeiro de 2016. 15

[71] Laura A. Seffino, Claudia Bauzer Medeiros, Jansle V. Rocha, e Bei Yi. Woodss - a spatial decision support system based on workflows. *Decision Support Systems*, 27(1-2):105–123, 1999. 19

[72] Amazon Web Services. Amazon s3. https://aws.amazon.com/pt/s3/. Acessado online em 03 de fevereiro de 2016. 51, 76

[73] Gurmeet Singh, Ewa Deelman, Gaurang Mehta, Karan Vahi, Mei Hui Su, G. Bruce Berriman, John Good, Joseph C. Jacob, Daniel S. Katz, Albert Lazzarini, Kent Blackburn, e Scott Koranda. The pegasus portal: web based grid computing. In *Proceedings of the 2005 ACM Symposium on Applied Computing (SAC), Santa Fe, New Mexico, USA, March 13-17, 2005*, pages 680–686, 2005. 23

[74] Munindar P. Singh e Mladen A. Vouk. Scientific workflows: Scientific computing meets transactional workflows. http://www.csc.ncsu.edu/faculty/mpsingh/papers/databases/workflows/sciworkflows.html. Acessado online em 12 de janeiro de 2016. 19

[75] Pivotal Software. Spring. https://spring.io. Acessado online em 18 de janeiro de 2016. 51

[76] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, e Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, August 2001. 33

[77] Andrew S. Tanenbaum e Maarten van Steen. *Distributed Systems: Principles and Paradigms (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006. 5

[78] Ian Taylor, Matthew Shields, Ian Wang, e Andrew Harrison. Visual grid workflow in triana. *Journal of Grid Computing*, 3(3):153–169, 2006. 23

[79] R. Thurlow. RPC: Remote Procedure Call Protocol Specification Version 2. RFC 5531 (Draft Standard), May 2009. 33

[80] Johns Hopkins University. Bowtie. http://bedops.readthedocs.org/en/latest/content/reference/file-management/conversion/sam2bed.html. Acessado online em 05 de fevereiro de 2016. 64

[81] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, e Maik Lindner. A break in the clouds: Towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008. 11, 14

[82] VMWare. Vmware. http://www.vmware.com/br. Acessado online em 11 de janeiro de 2016. 13

[83] E. White, L. McMillan, P. Romanski, M. O'Gara, e J. Bloomberg. Inter-cloud peering points. http://cloudcomputing.sys-con.com/node/1658700, 2010. xi, 16

[84] Wikipedia. Data access object. https://en.wikipedia.org/wiki/Data_access_object. Acessado online em 02 de fevereiro de 2016. 60

[85] Chee Shin Yeo, Rajkumar Buyya, Hossein Pourreza, M. Rasit Eskicioglu, Peter Graham, e Frank Sommers. Cluster computing: High-performance, high-availability, and high-throughput processing on a network of computers. In Albert Y. Zomaya, editor, *Handbook of Nature-Inspired and Innovative Computing*, pages 521–551. Springer, 2006. 6

[86] Zendesk. Zendesk. https://www.zendesk.com.br. Acessado online em 19 de janeiro de 2016. 15

[87] Qi Zhang, Lu Cheng, e Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, April 2010. xi, 7, 12, 14