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

BioNimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de Bioinformática

Hugo Vasconcelos Saldanha

Brasília
2012

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

BioNimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de Bioinformática

Hugo Vasconcelos Saldanha

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Orientadora
Prof.ª Dr.ª Maria Emília M. T. Walter

Coorientadora
Prof.ª Dr.ª Aletéia Patrícia F. Araújo

Brasília
2012

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

Coordenadora: Prof.ª Dr.ª Mylène Christine Queiroz de Farias

Banca examinadora composta por:

Prof.ª Dr.ª Maria Emília M. T. Walter (Orientadora) — CIC/UnB
Prof.ª Dr.ª Aletéia Patrícia F. Araújo (Coorientadora) — CIC/UnB
Prof.ª Dr.ª Alba Cristina M. A. de Melo — CIC/UnB
Prof. Dr. Vinod Rebello — IC/UFF

CIP — Catalogação Internacional na Publicação

Saldanha, Hugo Vasconcelos.

BioNimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de Bioinformática / Hugo Vasconcelos Saldanha. Brasília : UnB, 2012.
82 p. : il. ; 29,5 cm.

Dissertação (Mestrado) — Universidade de Brasília, Brasília, 2012.

1. computação em nuvem, 2. federação de nuvens híbrida,
3. bioinformática, 4. workflows

CDU 004

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

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

BioNimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de Bioinformática

Hugo Vasconcelos Saldanha

Dissertação apresentada como requisito parcial
para conclusão do Mestrado em Informática

Prof.ª Dr.ª Maria Emília M. T. Walter (Orientadora)
CIC/UnB

Prof.ª Dr.ª Aletéia Patrícia F. Araújo
CIC/UnB

Prof.ª Dr.ª Alba Cristina M. A. de Melo
CIC/UnB

Prof. Dr. Vinod Rebello
IC/UFF

Prof.ª Dr.ª Mylène Christine Queiroz de Farias
Coordenadora do Mestrado em Informática

Brasília, 22 de junho de 2012

Dedicatória

À Elisa, amada esposa, de todo o coração.

Agradecimentos

Agradeço, primeiramente, a minha orientadora, Prof.ª Dr.ª María Emília Machado Telles Walter, por sua dedicação e paciência em me ajudar neste projeto. Seu esforço em sempre buscar os melhores resultados e em ultrapassar supostos limites foi o que possibilitou a realização deste trabalho. Gostaria também de agradecer à Prof.ª Dr.ª Aletéia Patrícia Favacho Araújo, minha coorientadora, por sua disponibilidade constante em me auxiliar com seus conhecimentos. Os conceitos repassados a mim foram indispensáveis no direcionamento da pesquisa. E não poderia deixar de agradecer ao Prof. Dr. Vinod Rebello e à Prof.ª Dr.ª Alba Cristina M. A. de Melo pela gentileza de participar da minha banca examinadora.

Dedico um particular agradecimento aos colegas Edward Ribeiro e Carlos Borges. A ajuda e o companherismo durante a implementação do protótipo e a realização dos experimentos foram essenciais à execução do trabalho.

Por fim, sou eterna e especialmente grato a minha família — pais e irmãos —, a quem devo a formação de meu caráter e as condições em concluir mais essa etapa.

Resumo

O paradigma da Computação em Nuvem tem possibilitado o surgimento de um grande ecossistema composto por diferentes tecnologias e provedores de serviço com o objetivo de oferecer enorme quantidade de recursos computacionais sob demanda. Neste cenário, pesquisas científicas têm aproveitado a computação em nuvem como plataforma capaz de lidar com processamento e armazenamento em larga escala necessários na realização de seus experimentos. Em especial, a Bioinformática deve lidar com a grande quantidade de dados produzida pelas modernas máquinas de sequenciamento genômico. Neste contexto, várias ferramentas têm sido projetadas e implementadas para tirar proveito da infraestrutura oferecida pela computação em nuvem. Nuvens públicas, disponibilizadas por grandes provedores de serviço seriam capazes de oferecer, individualmente, recursos suficientes para atender ao poder computacional requerido pelas aplicações de bioinformática. Entretanto, esta escolha cria uma dependência tecnológica em relação ao provedor de serviço escolhido, tornando as instituições de pesquisa sujeitas às escolhas estratégicas deste provedor. Além disso, a infraestrutura computacional existente nessas instituições ficaria ociosa, ao invés de ser aproveitada em conjunto com o uso da nuvem pública. Como alternativa, surge a Federação de Nuvens Computacionais, que possibilita a utilização simultânea das diversas infraestruturas existentes nas várias instituições de pesquisa de maneira integrada, além de permitir a utilização dos recursos oferecidos pelas nuvens públicas. O presente trabalho tem como objetivo propor uma arquitetura de federação de nuvens computacionais híbrida, denominada BioNimbus, capaz de executar aplicações e *workflows* de bioinformática de maneira transparente, flexível, eficiente e tolerante a falhas, com grande capacidade de processamento e de armazenamento. Os serviços necessários à construção da federação são detalhados, juntamente com seus requisitos. Foi realizado um estudo de caso com um *workflow* e dados reais a partir da implementação de um protótipo da arquitetura, integrando nuvens públicas e privadas. Com os resultados obtidos, foi possível observar a real aplicabilidade de uma arquitetura de federação híbrida, em particular a BioNimbus, que atingiu as características projetadas inicialmente. Ao mesmo tempo, foram identificadas características que devem ser tratadas com o intuito de construir uma federação de nuvens computacionais híbrida que execute de forma eficiente e segura aplicações e *workflows* de bioinformática.

Palavras-chave: computação em nuvem, federação de nuvens híbrida, bioinformática, workflows

Abstract

The Cloud Computing paradigm has enabled the emergence of a large ecosystem composed of different technologies and service providers with the goal of providing enormous amount of computing resources on demand. In this scenario, scientists have taken advantage of cloud computing as a platform capable of handling the large scale processing and storage requirements to carry out their experiments. In particular, Bioinformatics must handle large amounts of data produced by modern genomic sequencing machines. Thus, several tools have been designed and implemented to take advantage of the infrastructure offered by cloud computing. However, as the computing power required can be very large, only public clouds, provided by large service providers, would be able to offer, individually, sufficient resources. In these conditions, there would be a technological dependence on the chosen service provider, making research institutions subject to the strategic choices of this provider. Furthermore, the existing computing infrastructure in these institutions would remain idle, causing great waste. Alternatively, Cloud Federation emerges as a way to allow the simultaneous use of several existing infrastructures in the various research institutions in an integrated manner, besides allowing the use of the resources offered by public clouds. The present work aims to propose an architecture of a hybrid cloud federation, called BioNimbus, capable of running applications and bioinformatics workflows in a transparent, flexible, efficient and fault-tolerant manner, with high processing power and huge storage capacity. The services required to build the federation are detailed, along with their requirements. We conducted a case study with a real workflow and real data through the implementation of a prototype of the architecture, integrating public and private clouds. With the results obtained, it was possible to observe the real applicability of the BioNimbus architecture, reaching the desired characteristics. At the same time, some details to be studied better in future work were identified in order to obtain a better implementation of a bioinformatics cloud federation.

Keywords: cloud computing, hybrid federated clouds, bioinformatics, workflows

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Um grande ecossistema composto por diferentes tecnologias e provedores de serviço [23, 36, 39, 57] surgiu graças ao advento do paradigma da computação em nuvem. Nesse ambiente, dispositivos de processamento e de armazenamento estão disponíveis de várias formas para que o usuário tenha a impressão de que esses recursos são virtualmente ilimitados.

Assim sendo, diversas pesquisas têm aproveitado o potencial computacional oferecido pelos provedores da computação em nuvem como uma plataforma capaz de lidar com o processamento e o armazenamento em larga escala. Nestas pesquisas, a computação em nuvem demonstrou ser, a partir dos resultados obtidos, uma plataforma promissora.

Entretanto, mesmo com o desenvolvimento contínuo da computação em nuvem e de sua utilização em diversas áreas, a quantidade de informação produzida por essas pesquisas, como informações genéticas, dados estatísticos, arquivos de imagens, entre outros, têm aumentado, ocasionando uma crescente necessidade de recursos de processamento e armazenamento. Assim, a manutenção de uma nuvem computacional privada que tenha recursos suficientes para atender a todas essas necessidade torna-se economicamente inviável devido aos custos associados, principalmente se houver escassez de recursos econômicos.

Uma alternativa seria a utilização de nuvens computacionais públicas, oferecidas por diversos provedores de serviço, como Amazon [57, 58], Google [39], Rackspace [73] e Microsoft [64]. Essas nuvens são capazes de colocar à disposição uma enorme capacidade computacional de processamento e de armazenamento aos seus usuários, com custos mais acessíveis, pois cada usuário paga por aquilo que usa, e o custo de manutenção da infraestrutura é dividido entre seus diversos clientes. Contudo, se somente fossem utilizadas nuvens públicas, haveria um desperdício de recursos computacionais já existentes nas instituições de pesquisa, pois suas infraestruturas (e.g. *clusters*) ficariam ociosas. Além disso, seria criada uma dependência tecnológica em relação ao provedor de serviço escolhido, sujeitando as instituições às escolhas estratégicas deste provedor. Essa dependência surge do fato de que cada provedor implementa, potencialmente, sua nuvem usando uma tecnologia diferente das tecnologias dos demais provedores. Isto ocorre pois não há padronização na implementação de um nuvem.

Nesse contexto, surgiu a federação de nuvens computacionais, que possibilita a utilização simultânea das diversas infraestruturas existentes nas várias instituições de pesquisa de maneira integrada, além de permitir a utilização dos recursos oferecidos pelas nuvens pú-

blicas quando os recursos das nuvens privadas integradas estiverem saturados. Ao mesmo tempo, arquiteturas e protocolos de federação têm sido propostos em diferentes trabalhos [15, 17]. Estas pesquisas demonstram a efetividade do uso da federação de nuvens computacionais como uma ferramenta para a otimização do uso de recursos de diferentes nuvens e para a eliminação da dependência de um único provedor de infraestrutura, colaborando ainda mais com a criação da impressão de recursos ilimitados disponíveis aos usuários.

Por sua vez, centenas de projetos genoma, por exemplo [31, 70, 86], têm criado enormes quantidades de informação, as quais precisam ser organizadas, armazenadas e gerenciadas com o objetivo de serem analisadas por dezenas de ferramentas computacionais, as quais requerem uma grande capacidade de processamento e armazenamento. Para analisar os dados produzidos utilizando as várias ferramentas disponíveis [3, 52, 55, 83] e para atender aos diversos tipos de estudos biológicos são montados diferentes *workflows*, criados pelos cientistas da computação com suporte dos projetos genoma.

Neste cenário, a federação de nuvens computacionais torna-se uma opção interessante para o controle e distribuição de processamento dos grandes volumes de dados produzidos por estes projetos.

1.1 Motivação

A federação de nuvens tem, recentemente, se mostrado uma técnica capaz de integrar diferentes infraestruturas, o que permite maior flexibilidade na escolha de provedores. Porém, ainda não existe uma proposta de arquitetura padrão para essas federações.

No âmbito específico da bioinformática, algumas ferramentas [82, 93] e *workflows* [4, 50, 51] foram implementados buscando tirar proveito do poder computacional da nuvem. Entretanto, durante o levantamento bibliográfico feito neste trabalho, notou-se que não existem propostas e implementações de arquiteturas de federação de nuvens computacionais visando a utilização de aplicações de bioinformática. Para dificultar ainda mais a situação, as ferramentas de bioinformática para a computação em nuvem até hoje implementadas não prevêem a possibilidade da utilização de federação e executam em um único ambiente, dificultando sua integração com diferentes nuvens.

1.2 Problema

Não existem arquiteturas de federação de nuvens computacionais para execução de *workflows* de bioinformática que:

- possibilitem a integração de diferentes ambientes de computação em nuvem de forma transparente ao usuário;
- sejam eficientes na distribuição do processamento das ferramentas de bioinformática disponíveis em diferentes instituições;
- sejam eficientes no armazenamento dos dados utilizados para execução dos *workflows*;
- e implementem tolerância a falhas.

1.3 Objetivos

1.3.1 Principal

O objetivo principal do presente trabalho é propor uma arquitetura para federação de nuvens computacionais capaz de executar aplicações e *workflows* de bioinformática comumente usados pelos diversos projetos genoma. A arquitetura proposta considera as seguintes características:

- Transparência, na visão do usuário, na integração de nuvens computacionais e das ferramentas de bioinformática oferecidas;
- Flexibilidade na integração de diferentes implementações de infraestruturas para nuvens computacionais e provedores;
- Flexibilidade na integração das várias ferramentas de bioinformática utilizadas por projetos genoma que venham a ser oferecidas como serviço nas diferentes infraestruturas integradas;
- Eficiência na distribuição da execução das ferramentas de bioinformática disponíveis nas infraestruturas, buscando tirar proveito dos recursos heterogêneos existentes de maneira otimizada;
- Eficiência no armazenamento dos dados utilizados na execução dos *workflows*, de forma que seja possível o uso de técnicas de localidade e replicação de dados para otimizar a transferência de dados entre a infraestrutura onde o dado foi armazenado e o local onde será utilizado, possibilitando a redução das transferências com consequente economia de largura de banda e redução do tempo total de execução do processamento;
- Tolerância a falhas, tanto na execução das ferramentas de bioinformática, quanto nos componentes que farão parte da arquitetura.

1.3.2 Específicos

Além da proposta de arquitetura, este trabalho tem como objetivos:

- Desenvolver uma implementação parcial da arquitetura proposta, integrando duas nuvens computacionais;
- Realizar um estudo de caso com dados biológicos reais utilizando um *workflow*;
- Discutir a execução do estudo de caso, verificando as vantagens e possíveis limitações da arquitetura proposta em relação a outras propostas de arquitetura para federação de nuvens computacionais presentes na literatura.

1.4 Descrição dos Capítulos

Este trabalho está dividido em mais seis capítulos. No Capítulo 2, são apresentados os conceitos de computação em nuvem e federação de nuvens computacionais. Além disso, são descritas algumas tecnologias usadas na computação em nuvem e detalhados os principais aspectos da federação de nuvens. Em seguida, algumas características necessárias e dificuldades a serem vencidas ao se propor uma arquitetura para federações são discutidas.

O Capítulo 3 contém uma breve introdução sobre Biologia Molecular, projetos genoma, ferramentas e *workflows*, tanto de uma maneira geral, quanto aqueles tipicamente usados em bioinformática. Ao fim deste capítulo são apresentados trabalhos que utilizam a computação em nuvem e a federação de nuvens computacionais.

Por sua vez, o Capítulo 4 apresenta a arquitetura proposta, denominada BioNimbus, com a descrição de seus componentes e a definição de como eles interagem entre si. Inicialmente, são apresentados e descritos os componentes da arquitetura BioNimbus. Em seguida, é apresentada a integração entre todos os componentes principais da arquitetura.

A arquitetura proposta servirá como referência para a implementação apresentada no Capítulo 5, no qual é descrito o estudo de caso em bioinformática, com os dados, *workflows* e ambiente utilizados, bem como detalhes desta implementação. Por fim, os resultados do estudo de caso são discutidos.

Finalmente, o Capítulo 6 conclui o presente trabalho e sugere trabalhos futuros.

Capítulo 2

Federação de Nuvens Computacionais

O objetivo deste capítulo é apresentar o paradigma da Computação em Nuvem e descrever as chamadas Federações de Nuvens Computacionais. Para isso, a Seção 2.1 apresenta conceitos básicos sobre computação em nuvem, mostrando como diversas pesquisas definiram esta tecnologia, quais são seus objetivos, como os serviços providos estão organizados e quem são seus provedores, e, por fim, qual a sua novidade em comparação às tecnologias de sistemas distribuídos consagradas. A Seção 2.2 apresenta o que vem a ser uma federação de nuvens computacionais, quais são as características específicas a este ambiente, e quais são os desafios a serem vencidos ao se implementar efetivamente uma federação de nuvens. Ao final da seção, duas propostas de arquitetura para implementação de federações de nuvens são descritas.

2.1 Computação em Nuvem

2.1.1 Conceitos Básicos

Computação em nuvem é um paradigma para provimento de infraestrutura computacional que vem criando um grande ecossistema formado por diferentes tecnologias e provedores de serviço, proporcionando aos usuários uma vasta variedade de opções de uso da nova tecnologia. Com ela, suas necessidades de processamento e armazenamento de dados são supridos de forma transparente.

Embora a computação em nuvem dependa de conceitos bem estabelecidos, tais como computação distribuída e virtualização, foi a entrada de grandes empresas, capazes de implantar enormes *datacenters* a custos competitivos que possibilitou a evolução e o desenvolvimento do paradigma. Armbrust *et al.* [5] definem a computação em nuvem como:

> **Definição 1** A união de aplicações oferecidas como serviço pela Internet com o *hardware* e o *software* localizados em *datacenters* de onde o serviço é provido.

Nesse esquema tem-se três tipos de atores: os provedores da infraestrutura (*hardware* e *software*), os usuários da infraestrutura (ou provedores de aplicações) e os usuários finais.

Contudo, essa definição ainda é muito abrangente, uma vez que pode ser confundida com a definição de outros paradigmas de computação distribuída bem conhecidos, como a Computação em *Grid* [34] e as Arquiteturas Orientadas a Serviço (SOA) [29]. Além disso, ela não explicita algumas características específicas da computação em nuvem, como a terceirização de serviços ou o modelo *pay-per-use*, os quais foram fatores importantes na diferenciação e adoção do paradigma.

De acordo com Foster *et al.* [35], a Computação em Nuvem pode ser definida como:

> **Definição 2** Um paradigma computacional altamente distribuído, direcionado por uma economia de escala, na qual poder computacional, armazenamento, serviços e plataformas abstratos, virtualizados, gerenciados e dinamicamente escaláveis são oferecidos sob demanda para usuários externos por meio da Internet.

Nesta definição estão expressos alguns pontos importantes, tais como a escalabilidade e a utilização sob demanda. Com ela, é possível perceber um grande número de características associadas ao novo paradigma. Vaquero *et al.* [92] colheram um grande conjunto de definições existentes na literatura, buscando formar uma definição com todas as características e uma outra com aquelas comuns a todas as fontes. Esta definição seria:

> **Definição 3** Nuvens são um grande *pool* de recursos virtualizados, facilmente utilizáveis. Os recursos podem ser reconfigurados dinamicamente de acordo com uma carga variável, permitindo uma utilização otimizada. Esse *pool* é explorado tipicamente por um modelo *pay-per-use* no qual garantias são oferecidas pelo provedor da infraestrutura, obedecendo um contrato de serviço.

Entretanto, características mínimas, que estivessem presentes em todas as demais definições estudadas, não foram encontradas. De qualquer forma, foi possível detectar as características mais frequentes, que seriam escalabilidade, modelo *pay-per-use* e virtualização.

Uma outra definição foi dada por Buyya *et al.* [16]:

> **Definição 4** A computação em nuvem é um modelo para oferta e utilização de recursos como serviços sob demanda, em um ambiente com múltiplos provedores, seguindo uma economia de escala.

Esta definição estabelece que o objetivo da computação nuvem é proporcionar a idéia da existência ilimitada de recursos para utilização imediata por seus usuários, os quais pagam apenas pelos recursos efetivamente usados (modelo *pay-per-use*), sem haver necessidade de gerenciamento direto da infraestrutura do provedor de serviços. Assim, usuários e desenvolvedores de aplicações, ao utilizarem a computação em nuvem, não precisam mais se preocupar com os gastos para montar a infraestrutura de *hardware* ou com a despesa de pessoal para operá-la ao tentar armazenar grandes quantidades de dados ou oferecer alguma nova aplicação na Internet. Além disso, a computação em nuvem oferece elasticidade suficiente para que a infraestrutura virtual cresça proporcionalmente à utilização

do serviço disponibilizado pela aplicação desenvolvida, evitando assim prejuízos por custo excessivo ou por perda de clientes causados por mal dimensionamento da infraestrutura.

Neste contexto, no qual existem diversas definições, e com a intenção de colaborar no debate sobre computação em nuvem e sua definição, casos de uso, tecnologias, problemas, riscos e benefícios, o *National Institute of Standards and Technology* (NIST) dos Estados Unidos propôs também uma definição para o paradigma [63]:

> **Definição 5** Computação em nuvem é um modelo para permitir acesso de rede fácil e ubíquo para um *pool* de recursos computacionais configuráveis (por exemplo, redes, servidores, armazenamento, aplicativos e serviços) que podem ser rapidamente fornecidos e liberados com esforço de gerenciamento e interação com o provedor de serviço mínimos.

2.1.2 Arquitetura

Há diferentes propostas de arquitetura apresentadas na literatura [14, 35, 53, 63, 96], as quais devem ser analisadas e comparadas.

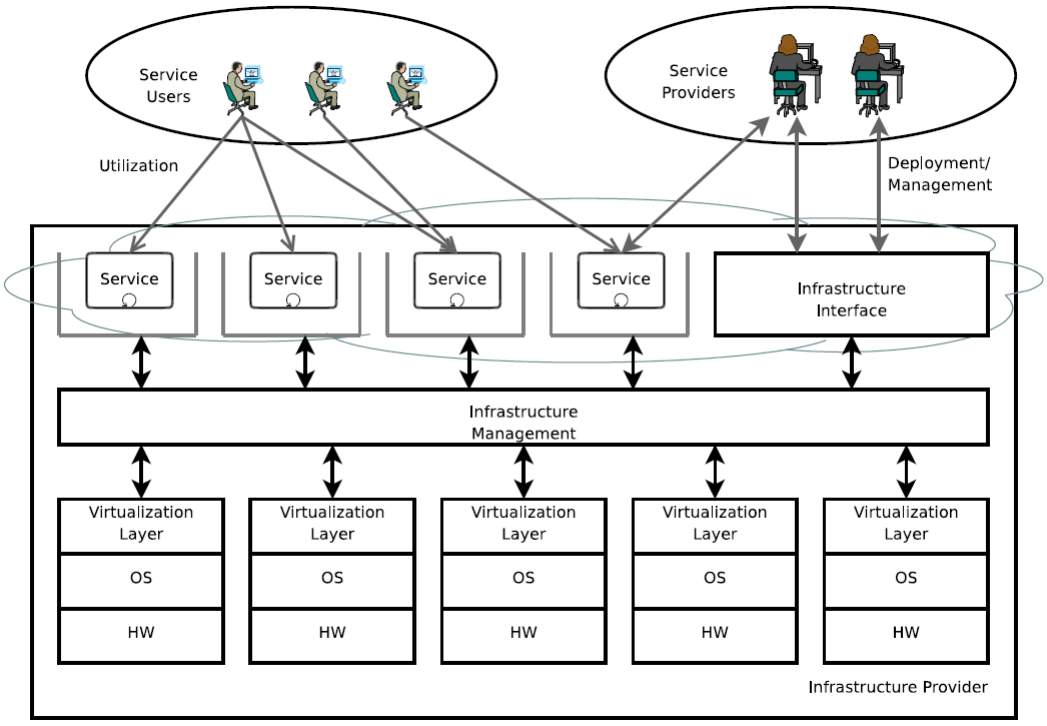

Figura 2.1: Atores envolvidos na computação em nuvem e sua interação [92].

No intuito de se ter uma arquitetura de referência para a computação em nuvem, um primeiro passo é identificar os atores envolvidos e como eles interagem entre si. O primeiro grupo de atores são os **provedores de serviço**. Eles oferecem serviços baseados em *software* por meio de interfaces disponíveis pela Internet, que são acessadas pelos **usuários**. O objetivo da computação em nuvem é retirar a responsabilidade do provedor de serviço de criar e manter a infraestrutura necessária para hospedar seus serviços. Para

realizar esse papel, surgem os **provedores de infraestrutura**, responsáveis por gerir os recursos computacionais, para que os provedores de serviço ganhem flexibilidade e reduzam custo de manutenção [92]. A Figura 2.1 reflete essa nova disposição de recursos, serviços e atores possibilitada pela computação em nuvem.

Os serviços mapeados acima podem ser implantados de diferentes formas pelos provedores. Os modelos de implantação existentes [63] para a computação em nuvem são:

- **Nuvem privada**: a infraestrutura da nuvem é operada para uso de uma única organização. Pode ser gerenciada pela própria organização ou por terceiros e pode estar implantada interna ou externamente à organização.
- **Nuvem comunitária**: a infraestrutura da nuvem é compartilhada por várias organizações e serve como ferramenta para um grupo específico de usuários com interesses em comum — missão, resquisitos ou políticas de uso. Pode ser implantada interna ou externamente às organizações envolvidas.
- **Nuvem pública**: a infraestrutura da nuvem é disponibilizada para o público em geral ou para um grande grupo corporativo, e pertence a uma organização que vende serviços.
- **Federação de nuvens ou nuvem híbrida**: a infraestrutura da nuvem é a composição de duas ou mais nuvens (privadas, comunitárias ou públicas) que permanecem entidades separadas, mas são unidas por tecnologia padronizada ou proprietária que permite a portabilidade de dados e aplicações para, por exemplo, balanceamento de carga entre as diferentes nuvens.

Um segundo passo necessário para se analisar a computação em nuvem é realizar o mapeamento dos vários tipos de tecnologias utilizadas com seus papéis dentro da arquitetura geral da computação em nuvem. Esta arquitetura (veja Figura 2.2) divide-se em três camadas que organizam os diferentes tipos de serviços oferecidos [35]. Essas camadas são:

- ***Infrastructure-as-a-Service*** (IaaS): provê serviços que oferecem recursos computacionais ao usuários, tais como virtualização de nós computacionais, armazenamento de dados e redes virtuais;
- ***Platform-as-a-Service*** (PaaS): provê serviços que oferecem aos usuários ambientes de programação e execução de aplicações distribuídas;
- ***Software-as-a-Service*** (SaaS): provê aplicações oferecidas aos usuários como serviço.

As aplicações oferecidas como serviço na camada SaaS podem ser desenvolvidas ou executadas pelas plataformas da camada PaaS, ou utilizar diretamente os recursos oferecidos pela camada IaaS. Os usuários usufruem dos serviços de todas as camadas remotamente, por meio da Internet.

A seguir, são descritas em mais detalhes as três camadas que compõem a arquitetura da computação em nuvem.

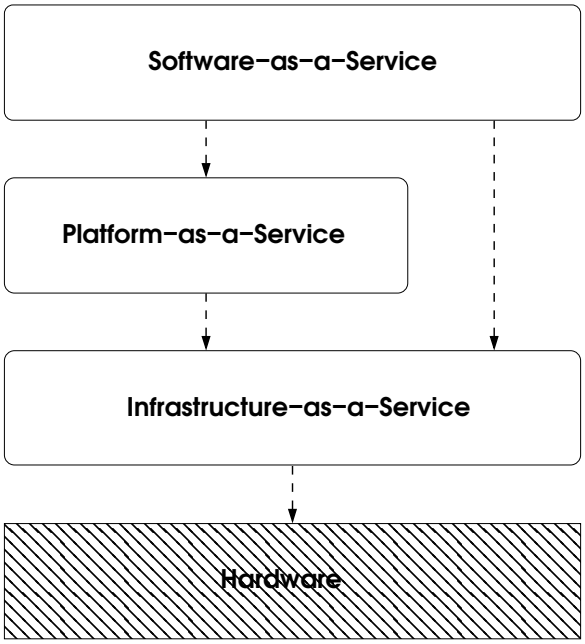

Figura 2.2: A arquitetura em camadas da computação em nuvem.

Infrastruture-as-a-Service

Na camada *Infrastructure-as-a-Service* encontram-se os serviços que oferecem recursos computacionais, disponibilizados pelos provedores de infraestrutura. Nela, os recursos se assemelham muito com uma infraestrutura física e é possível ter o controle de quase toda a pilha de *software* utilizada. Entre os recursos disponíveis estão virtualização de nós computacionais, armazenamento de dados, redes de transmissão de dados, sistemas de arquivos distribuídos e modelos computacionais para paralelismo e distribuição de processamento.

Para atender aos preceitos da computação em nuvem, uma interface de gerenciamento é oferecida aos serviços das camadas superiores para que seja possível a automação do processo de inicialização de nós computacionais, configuração de rede de dados, definição de capacidades de armazenamento, configuração de parâmetros de tolerância a falha, entre outros. Esta interface de gerenciamento é utilizada tanto por provedores de serviço como por usuários comuns que desejam fazer uso diretamente dos recursos computacionais. Quando acessado por usuários comuns, o próprio provedor de infraestrutura realiza o papel de provedor de serviço. Alguns exemplos de tecnologias existentes na camada *IaaS* são:

- *OpenNebula* [84]: gerenciador de recursos físicos e virtuais;
- *GoogleFS* [37]: sistema de arquivos distribuídos;
- *Apache Hadoop* [36]: implementação do modelo computacional MapReduce [23] desenvolvido pela Google;
- *Amazon Dynamo* [24]: banco de dados distribuído.

Um provedor de infraestrutura da camada *IaaS* que se destaca é a Amazon com o *Elastic Compute Cloud (EC2)* [57] e com o *Simple Storage Service (S3)* [58]. Por meio dos serviços oferecidos, outros provedores e usuários comuns são capazes de ter milhares de nós computacionais e utilizar vários *gigabytes* de capacidade de disco com grande flexibilidade e baixo custo.

Platform-as-a-Service

Na camada *Platform-as-a-Service* estão serviços que oferecem plataformas de programação e de execução. Para isso, os provedores destes serviços utilizam os serviços disponibilizados pelos provedores de infraestrutura.

Com estas plataformas é possível desenvolver aplicações específicas para a nuvem, utilizando APIs fornecidas pelos provedores dos serviços, e utilizar o ambiente em nuvem para executar aplicações customizadas. Sendo assim, esta camada permite uma escalabilidade impressionante a desenvolvedores e empresas que oferecem serviços por meio de aplicações *web*, aumentando a utilização de recursos de maneira transparente e quase instântanea, de acordo com a utilização do serviço, sem a necessidade de trabalho com aquisição, instalação e configuração de novo *hardware*, e com a instalação e configuração da aplicação. Exemplos de ambientes da camada PaaS são:

- *Google App Engine* [39]: uma plataforma para desenvolvimento e disponibilização de aplicações *web*;
- *Windows Azure* [64]: uma plataforma para execução de aplicações que também oferece diferentes ferramentas de programação.

Software-as-a-Service

Na camada *Software-as-a-Service (SaaS)* encontram-se as aplicações desenvolvidas especificamente para o ambiente da computação em nuvem, que são fornecidas como serviço por provedores para usuários comuns. Os serviços desta camada podem usar os ambientes de programação e execução fornecidos pelos provedores da camada *PaaS*, ou podem utilizar diretamente a infraestrutura oferecida pelos provedores da camada *IaaS*.

O usuário tem acesso remoto a essas aplicações, as quais podem ser acessadas a qualquer momento, podendo haver cobrança baseada na utilização do serviço, em substituição a aplicações que executem localmente no *hardware* do usuário. Como vantagem, as tarefas de manutenção, de operação e de suporte são repassadas para o provedor do serviço. Exemplos de aplicações desta camada são:

- *Google Docs* [40]: um conjunto de aplicações de escritório oferecidas na *web*;
- *Sales Cloud* [80]: o sistema de CRM (*Customer Relationship Manager*) oferecido pela *Salesforce*, na qual executivos e vendedores da empresa contratante do serviço podem gerenciar clientes, vendas e contatos diretamente na *web*.

2.1.3 Comparação com Computação em *Grid*

A definição clara de computação em nuvem se faz necessária para que se evitem possíveis confusões de conceitos com outras tecnologias semelhantes. A mais conhecida delas é a computação em *grid*, tecnologia bem difundida entre os pesquisadores e bastante utilizada em diversos tipos de pesquisas científicas.

O pesquisador Ian Foster [34] definiu a computação em *grid* a partir de três pontos primordiais da seguinte maneira:

> **Definição 6** Um *grid* é um sistema que *a)* coordena recursos que não estão sujeitos a controle centralizado, *b)* utilizando interfaces e protolocos padronizados, abertos e de uso geral, *c)* para fornecer *quality-of-service* (QoS) não trivial.

Tanto a computação em nuvem como a computação em *grid* são tecnologias que tiram proveito da distribuição do processamento em plataformas distribuídas tais como *clusters* computacionais, atingindo alto grau de paralelismo a depender da escala da plataforma e do tipo de algoritmo utilizado. Entretanto, pesquisadores apontaram, recentemente, algumas diferenças conceituais e práticas, tomando essas diferenças como possíveis causas para a grande adoção da computação em nuvem, tanto no mundo acadêmico quanto no comercial, em detrimento da computação em *grid*.

O mesmo Ian Foster apontou o que é comum e o que difere nas duas tecnologias [35]. Entre as diferenças, podem ser citadas:

- Na computação em nuvem, os recursos são contratados de acordo com a utilização efetiva, como ciclos de CPU ou *bytes* de disco, ao contrário de *grids*, onde os recursos normalmente são alocados por projetos, a partir de uma previsão;
- Devido ao uso de virtualização e o consequente isolamento de ambientes, a computação em nuvem permite que aplicações de vários usuários executem simultaneamente, diferentemente do que normalmente acontece em *grids*, na qual a execução das aplicações obedecem uma fila, onde aguardam por recursos disponíveis;
- Também por meio da virtualização, a computação em nuvem permite ao usuário criar o ambiente que mais se ajuste ao seu caso de uso, seja aumentando o tamanho do armazenamento para aplicações que processem muitos dados, sem a necessidade de CPUs poderosos, seja utilizando máquinas com bastante memória RAM, sem ser preciso discos com grandes capacidades, otimizando o uso dos recursos computacionais. Já a computação em *grid* não tem aproveitado essa vantagem da virtualização;
- Os modelos de programação disponíveis na computação em nuvem são mais simples e transparentes, enquanto os de *grid* obedecem os modelos tradicionais de computação paralela e distribuída, como MPI (*Message Passing Interface*) [33], nos quais é preciso se ater a problemas de heterogeneidade do ambiente, diferentes domínios administrativos, entre outros.

2.2 Federação de Nuvens Computacionais

A computação em nuvem tem buscado atingir níveis cada vez melhores de eficiência na disponibilização de serviços. Como foi apresentado nas seções anteriores, os serviços prestados por uma nuvem variam entre infraestruturas, plataformas e *software*. Seus clientes podem ser desde um usuário simples a outras nuvens, passando por instituições acadêmicas e grandes empresas. Além das grandes nuvens públicas, mantidas por grandes organizações, centenas de outras nuvens menores, privadas ou híbridas, vêm sendo implantadas de maneira heterogênea e independente. Com isso, surge o cenário onde a federação de nuvens computacionais interoperáveis se torna uma alternativa interessante para otimizar o uso dos recursos oferecidos por essas diversas instituições.

2.2.1 Conceitos Básicos

Bittman [11] afirma que a evolução do mercado da computação em nuvem pode ser dividida em três fases, sendo que a terceira é a atual. Na fase 1 (monolítica), serviços de computação em nuvem são baseados em arquiteturas proprietárias ou são oferecidos por megaprovedores, como por exemplo Google, Microsoft, Amazon e Salesforce.

Na fase 2 (cadeia vertical de fornecimento), alguns provedores de serviços tiram proveito de serviços oferecidos por outros provedores de nuvens. Um exemplo são empresas fabricantes de *software* movendo suas aplicações para a camada SaaS sobre plataformas de terceiros, como a Microsoft Azure [64] e o Google App Engine [39]. Neste cenário, os ambientes de computação em nuvem ainda são proprietários e isolados, mas se inicia a construção da integração entre nuvens.

Por fim, na fase 3 (federação horizontal), pequenos provedores federam-se horizontalmente para atingir maior escalabilidade e eficiência no uso de seus recursos. Com isso, projetos tiram proveito da federação para alargar suas capacidades, surgem mais escolhas em cada uma das camadas da nuvem, e iniciam-se discussões sobre padrões de interoperabilidade e federação.

Assim sendo, a federação de nuvens computacionais, também chamada de *inter-cloud* ou *cross-cloud* [17], é uma área de pesquisa particular em computação em nuvem e pode ser definida como um conjunto de provedores de nuvens computacionais, públicos e privados, conectados por meio da Internet. Entre seus objetivos, estão listados [15, 17]:

- Alcançar de maneira mais efetiva a impressão de que existem recursos ilimitados disponíveis para uso;
- Permitir a eliminação da dependência de um único provedor de infraestrutura;
- Otimizar o uso dos recursos dos provedores federados.

Para atingir os objetivos acima, a federação permite a cada operador de nuvem computacional aumentar sua capacidade de processamento e armazenamento ao requisitar mais recursos às demais nuvens da federação. Consequentemente, o operador é capaz de satisfazer requisições de usuários que sejam feitas após a saturação dos recursos de sua nuvem computacional, recursos ociosos dos outros provedores são aproveitados e, caso um provedor esteja fora do ar, pode-se requisitar recursos a um outro.

2.2.2 Requisitos e Desafios

Apesar das vantagens óbvias da federação de nuvens computacionais, sua implementação não é de modo algum trivial, pois nuvens têm características específicas e, por isso, modelos tradicionais de federação não são aplicáveis. Normalmente, os modelos tradicionais são baseados em acordos prévios feitos pelos membros da federação. Isso é possível pois, nesses ambientes tradicionais, os recursos são estáticos e pouco heterogêneos. Ao contrário, no cenário de computação em nuvem, os recursos são muito heterogêneos e altamente dinâmicos, impossibilitando esse tipo de acordo [17].

Dada esta situação, para que seja possível a criação de uma federação de nuvens computacionais, é necessário atender aos seguintes requisitos [15, 17]:

- **Automatismo**: uma nuvem membro da federação, usando mecanismos de descoberta, deve ser capaz de identificar as demais nuvens da federação e quais são seus recursos, reagindo a mudanças de maneira transparente e automática.

- **Previsão de carga de aplicações**: o sistema que implementa a federação deve possuir alguma forma de prever as demandas e comportamentos dos serviços oferecidos, tal que consiga, de maneira eficiente e dinâmica, escalonar sua execução entre os provedores participantes da federação.

- **Mapeamento de serviços a recursos**: os serviços oferecidos pela federação devem ser mapeados aos recursos disponíveis de maneira flexível para que se consiga atingir os melhores níveis de eficiência, custo-benefício e utilização. Em outras palavras, o escalonamento da execução deve computar a melhor combinação *hardware-software* de forma a garantir a qualidade do serviço e o menor custo, levando em conta a incerteza da disponibilidade dos recursos.

- **Modelo de segurança interoperável**: a federação deve permitir a integração de diferentes tecnologias de segurança, fazendo com que as nuvens membros não necessitem mudar suas respectivas políticas de segurança ao entrar na federação.

- **Escalabilidade no monitoramento de componentes**: dada a possível grande quantidade de participantes, a federação deve ser capaz de lidar com as várias filas de trabalho e o grande número de requisições, de forma que consiga manter o gerenciamento dos diversos componentes da federação sem perder em escalabilidade e desempenho.

2.2.3 Propostas de Arquiteturas

Na tentativa de guiar a implementação de uma federação de nuvens computacionais de modo que os requisitos acima mencionados sejam atingidos, algumas propostas de arquitetura são apresentadas na literatura.

Em uma delas, Celesti *et al.* [17] propuseram um ambiente para federação no qual existem dois tipos de nuvens computacionais: nuvem local e nuvem estrangeira. A nuvem local é o provedor que está com sua infraestrutura saturada e, consequentemente, repassa requisições para as nuvens estrangeiras. Estas, por sua vez, são os provedores que concedem o uso de seus recursos ociosos à nuvem local, com ou sem cobrança. Um provedor

pode ser uma nuvem local e uma nuvem estrangeira ao mesmo tempo. Os usuários fazem então requisições de uso diretamente a uma das nuvens da federação e esta, agindo como uma nuvem local, repassaria requisições às demais se porventura não conseguisse suprir as necessidades dos usuários.

Nesta arquitetura, os pesquisadores propõem que em cada uma das infraestruturas disponibilizadas na federação exista um gerenciador, chamado de *Cross-Cloud Federation Manager* (CCFM). O CCFM seria então responsável por realizar as operações necessárias para que o estabelecimento da federação seja possível, e por atender os chamados trocados entre as nuvens da federação. Considerando a dinamicidade do ambiente da federação, no qual nuvens com diferentes recursos e diferentes mecanismos de segurança aparecem e desaparecem, são propostos diferentes agentes que compõem o CCFM, que realizariam, cada um, um passo do processo de atendimento da requisição dos usuários. O *Discovery Agent* seria responsável por identificar quais nuvems fazem parte da federação e quais seus recursos. O *Match-Making Agent* faria a escolha de quais nuvens seriam as melhores para atender uma determinada requisição de usuário. Finalmente, o *Authentication Agent* criaria o canal de segurança entre a nuvem local e a nuvem estrangeira, de forma que a primeira seja capaz de usar os recursos da última de acordo com as políticas de segurança desta. A Figura 2.3 descreve graficamente esta proposta.

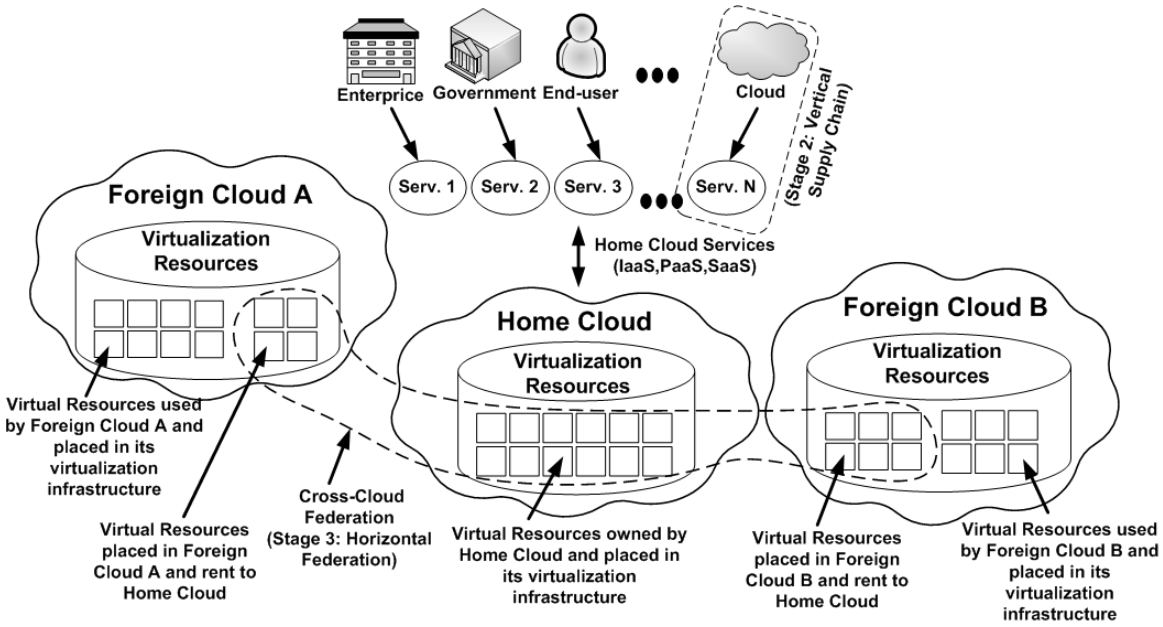

Figura 2.3: Proposta de arquitetura para federação por Celesti *et al.* [17].

Visando, também, uma arquitetura que atenda aos requisitos existentes para a implementação de uma federação de nuvens computacionais, Buyya *et al.* [15] apresentaram uma proposta alternativa. Nela, ao contrário da proposta anterior, o usuário não interage diretamente com um componente da arquitetura disponível na infraestrutura de um provedor, mas utiliza um componente externo, chamado de *Cloud Broker* (CB). Ele é responsável por criar a comunicação entre o usuário e a federação de nuvens, e por identificar quais provedores possuem recursos disponíveis para atender os requisitos de qualidade de serviço (QoS) exigidos. Uma vez identificada a(s) infraestrutura(s) para execução, ele também é responsável por fazer a submissão da tarefa desejada. Para conseguir os dados

necessários sobre as nuvens disponíveis na federação o CB consulta outro componente da arquitetura chamado *Cloud Exchange* (CEx). Este funciona como um registro a ser consultado pelos CBs, com informações sobre as infraestruturas, tais como custos de utilização, recursos disponíveis e padrões de execução. Além disso, o CEx oferece serviços para mapeamento de requisições dos usuários a provedores que melhor as atenderiam. Por fim, em cada uma das infraestruturas oferecidas pelos provedores, a exemplo da proposta anterior, existe um componente da arquitetura chamado de *Cloud Coordinator* (CC), responsável por incluir a infraestrutura na federação e expor os recursos disponíveis aos usuários da federação. Para atender as requisições dos usuários, o CC implementa a autenticação e o estabelecimento de um acordo de qualidade de serviço com cada CB, e escalona as requisições para a infraestrutura de acordo com esta negociação. Em adição, o CC identifica os recursos disponíveis na infraestrutura e monitora sua utilização para informar estes dados ao CEx, juntamente com os dados de custo. A Figura 2.4 descreve esquematicamente a arquitetura proposta.

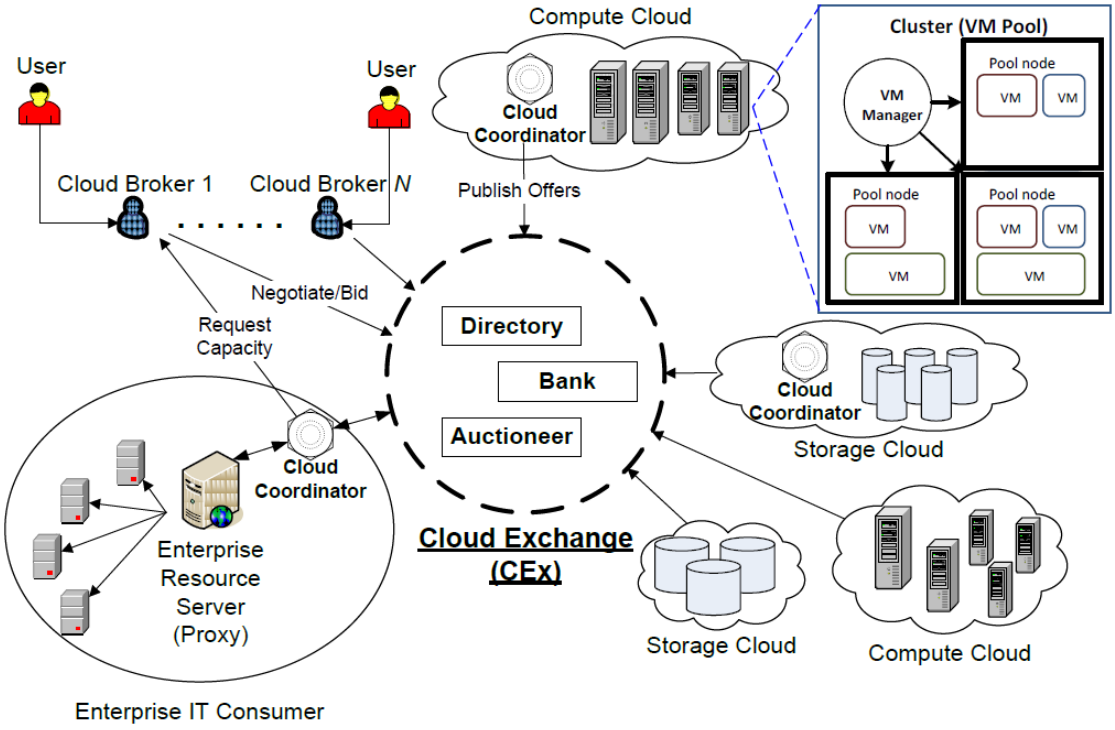

Figura 2.4: Proposta de arquitetura para federação por Buyya *et al.* [15].

Capítulo 3

Workflows em Bioinformática

Neste capítulo, são descritos de forma breve os projetos genoma, em particular as ferramentas que compõem *workflows* de bioinformática aplicados durante esses projetos. Primeiramente, na Seção 3.1 são apresentados conceitos básicos de Biologia Molecular. Em seguida, a Seção 3.2 descreve como é realizado o sequenciamento de material genético e quais são as características dos dados produzidos neste processo, além de oferecer uma amostra das condições necessárias para seu armazenamento em ambientes computacionais. Por último, na Seção 3.3 são descritos os *workflows* computacionais e como eles são aplicados na bioinformática. No decorrer da seção são apresentadas as fases que formam os *workflows* de projetos genoma, com exemplos de ferramentas de bioinformática e de sistemas de gerenciamento de *workflows*.

3.1 Conceitos Básicos em Biologia Molecular

A Biologia Molecular é uma área do conhecimento na qual são estudados os processos celulares relacionados à síntese de proteínas a partir de informações genéticas, as quais estão contidas nos ácidos nucleicos (DNA e RNA). Trata-se de um campo de estudo diretamente relacionado à genética e à bioquímica.

O DNA e o RNA são macromoléculas informacionais, constituídas por subunidades em sequência linear ordenada denominadas nucleotídeos. Já as proteínas são macromoléculas que desempenham variadas funções nos organismos, e suas subunidades são os aminoácidos [1, 65].

O **DNA** (*DesoxyriboNucleic Acid*) ou ácido desoxirribonucléico é uma molécula no formato de dupla hélice composta por duas longas fitas complementares de nucleotídeos que são mantidas unidas por pontes de hidrogênio entre os pares de base Guanina (G) – Citosina (C) e Adenina (A) – Timina (T). Armazenamento e transmissão de informações são as únicas funções conhecidas do DNA. Este codifica a informação por meio da sequência de nucleotídeos ao longo da fita. Cada base, A,C,T ou G, pode ser considerada como uma das letras de um alfabeto de quatro letras que significam mensagens biológicas na estrutura química do DNA. Nos organismos eucariotos, isto é, aqueles que possuem em suas células um núcleo delimitado, o DNA está localizado justamente no núcleo celular. O DNA nuclear é dividido em uma série de diferentes cromossomos. Cada cromossomo consiste de uma única e enorme molécula de DNA linear com proteínas associadas que dobram e empacotam a fita fina de DNA em uma estrutura mais compacta. Assim,

um cromossomo é formado de uma longa molécula de DNA que contém inúmeros genes organizados linearmente [1].

Por sua vez, o **RNA** (*RiboNucleic Acid*) ou ácido ribonucléico é uma molécula composta por uma fita simples formada pela sequência de nucleotídeos que contêm as bases Guanina (G), Citosina (C), Adenina (A) e Uracila (U), sendo esta última divergente em relação ao DNA [65]. Os RNAs possuem uma variedade de funções e, nas células, são encontrados como diversas classes. Os RNAs ribossômicos (rRNA) são componentes estruturais dos ribossomos, unidades celulares que realizam a síntese de proteínas. Os RNAs mensageiros (mRNA) são intermediários que transportam a informação genética de um ou de poucos genes até os ribossomos, onde as proteínas correspondentes podem ser sintetizadas. Os RNAs transportadores (tRNA) são moléculas adaptadoras que traduzem a informação presente no mRNA em uma sequência específica de aminoácidos [1].

As **proteínas** são as macromoléculas mais abundantes das células vivas, sendo constituídas por sequências ordenadas de aminoácidos. As células contém milhares de proteínas diferentes, cada uma com uma atividade biológica distinta [65].

Nos organismos vivos, a unidade fundamental da informação genética é o **gene** [1]. Bioquimicamente, é definido como um segmento de DNA, ou em alguns casos de RNA, que codifica a informação necessária para a produção de um determinado composto biológico funcional [65]. Por sua vez, **expressão gênica** é definida como o processo por meio do qual a célula traduz a sequência nucleotídica de um gene na sequência de aminoácidos de uma proteína. Ainda, a série completa de informações do DNA de um organismo é chamada **genoma**, e este contém a informação para todas as proteínas que o organismo irá sintetizar.

Conforme ressaltado, a informação genética é mantida em uma sequência linear de nucleotídeos no DNA. A duplicação da informação genética ocorre pelo uso de uma fita de DNA como molde para a formação da fita complementar, em um processo denominado **replicação**. A informação genética é lida e processada em duas etapas. Na **transcrição**, os segmentos de uma sequência de DNA são usados para guiar a síntese de moléculas de RNA, os mRNA. Quando a célula necessita de uma proteína específica, a sequência de nucleotídeos da porção apropriada de uma longa molécula de DNA em um cromossomo é primeiramente copiada sob a forma de RNA, os tRNA. Em seguida, na **tradução**, tais cópias de RNA de segmentos de DNA são usadas diretamente como molde para direcionar a síntese da proteína.

Portanto, pode-se afirmar que o fluxo de informação genética nas células é de DNA para RNA, e deste para proteína. Todas as células dos organismos vivos, desde bactérias até seres humanos, expressam sua informação genética dessa maneira, sendo este um princípio fundamental denominado o **dogma central da biologia molecular**. A Figura 3.1 representa graficamente as moléculas e o fluxo de informação genética.

Quanto às características dos genes de organismos eucariotos, estes são encontrados sob a forma de pequenos pedaços de sequências codificantes, denominadas sequências expressas ou éxons. Estes éxons são intercalados por sequências longas, as sequências intervenientes ou íntrons. Nesse sentido, a porção codificante de um gene eucariótico é, em geral, apenas uma pequena fração do comprimento do gene. No caso dos organismos procariotos – isto é, que não possuem núcleo delimitado em suas células – os genes consistem de uma porção contínua de DNA codificante que é diretamente transcrita em mRNA [1].

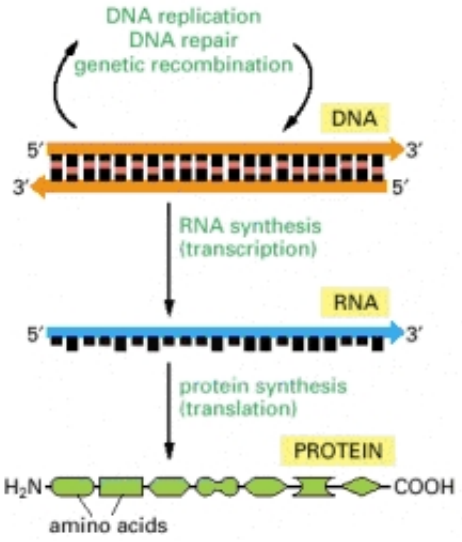

Figura 3.1: Fluxo de informação genética DNA-RNA-proteína [1].

Sendo assim, existem variações de como a informação flui do DNA para a proteína. A principal destas é que os transcritos de RNA, produzidos a partir da transcrição de DNA e chamados de **transcriptoma**, em células eucarióticas são submetidas a uma série de etapas de processamento no núcleo. Dentre estas etapas está o *splicing* de RNA, processo no qual as sequências dos íntrons são removidas dos RNA sintetizados a partir do DNA. Nos eucariotos existe também um mecanismo denominado *splicing* alternativo de RNA, no qual há produção de proteínas diferentes a partir de um mesmo transcriptoma, devido ao *splicing* processado de maneiras distintas [1].

3.2 Projetos Genoma

Os avanços técnicos da Biologia Molecular têm facilitado o estudo das células e de suas macromoléculas, provendo novas ferramentas seja para determinar a função de proteínas ou para identificar genes. De maneira geral, tais ferramentas envolvem o isolamento, a clonagem e o sequenciamento de DNA.

O sequenciamento é a tarefa de obter os nucleotídeos que compõem fragmentos de sequências pertencentes ao DNA ou ao RNA de um ou mais organismos em um projeto genoma, dependendo dos objetivos de cada projeto. Estes fragmentos são também chamados de *short-read-sequences* (SRS).

Um projeto genoma é desenvolvido em geral por uma equipe multidisciplinar, composta por biólogos nos laboratórios de biologia molecular e por cientistas da computação nos laboratórios de bioinformática. Existem diferentes tipos de foco nos projetos genoma. Alguns exemplos são:

- Reconstrução do genoma de um organismo (projetos genoma);
- Obtenção dos transcritos de RNA (transcriptoma);
- Estudo de material genético coletado de amostras do ambiente (epigenética);

- Pesquisa de mudanças na expressão genética de tecidos afetados por doenças (genes diferencialmente expressos).

Em um projeto de sequenciamento, primeiramente é produzido um grande número de SRS (fragmentos de DNA ou RNA), por meio de sequenciadores. Em seguida, elas são convertidas para sequências de caracteres (*strings*) formadas pelas letras **A**, **C**, **G** e **T** ou **U**, cada uma correspondendo a uma das bases nitrogenadas de DNA ou RNA. Essa primeira parte do projeto é realizada nos laboratórios de biologia molecular. As demais fases são todas realizadas em um laboratório de bioinformática, que centraliza o armazenamento, o gerenciamento e o processamento dos dados gerados pelo projeto.

O sequenciamento pode ser realizado automaticamente por meio de máquinas [81], chamadas sequenciadores automáticos, e vem obtendo grandes avanços graças às novas tecnologias desenvolvidas por empresas como Illumina [44], Applied Biosystems [88] e 454 Life Sciences [21].

O sequenciador ***Illumina*** utiliza a técnica de sequenciamento por síntese, onde uma base é incorporada por vez à sequência sendo determinada. No processo de amplificação do fragmento de DNA original, são gerados vários grupos de sequências, cada grupo com aproximadamente 1 milhão de cópias do fragmento original. Com o material genético amplificado é realizado o sequenciamento em si. Nele, através de reações químicas, um nucleotídeo é adicionado a cada sequência por vez, repetidamente, até se chegar a uma SRS de aproxidamente 32 bases de comprimento. Ao fim, o sequenciamento pode chegar a gerar mais de 1 bilhão de bases sequenciadas [25].

Por outro lado, o sequenciador ***454*** utiliza a técnica de sequenciamento conhecida como pirosequenciamento. Nesta técnica, cada nucleotídeo incorporado a uma fita do DNA sendo sequenciado, por meio de reações químicas, libera luz de intensidades diferentes para cada tipo de nucleotídeo. Um sensor é utilizado para determinar as bases que formam o fragmento de DNA sequenciado. O processo de amplificação e sequenciamento deste sequenciador gera milhões de SRS de aproximadamente 250 bases.

Para dar uma amostra do volume de dados obtidos em projetos genoma reais são apresentados dois exemplos de estudo:

- Filichkin *et al.* [31] produziram e trabalharam com aproximadamente 271 milhões de SRS, cada uma delas com 32 pares de bases nitrogenadas, com o objetivo de identificar *splicing* alternativos da planta *Arabidopsis thaliana*. Este organismo possui um genoma relativamente pequeno, de aproximadamente 120 milhões de pares de bases. Dessa forma, foram produzidos aproximadamente 17,3 GB de dados no sequenciamento, os quais foram mapeados a mais ou menos 240 MB de dados;

- Sultan *et al.* [86] também tinham o objetivo de identificar *splicing* alternativos, mas no genoma humano. Para isso, trabalharam com cerca de 15 milhões de SRS e os 3 bilhões de bases nitrogenadas do genoma humano completo. Isso equivale a mapear de 960 MB a 6 GB.

Todas as SRS produzidas pelo sequenciamento são, então, submetidas a análises realizadas por ferramentas computacionais. Essas análises são necessárias pois os fragmentos produzidos pelos sequenciadores automáticos devem ter sua qualidade verificada (pode ter havido erros de laboratório ou de sequenciamento), ser agrupados se os fragmentos

forem muito pequenos ou ter identificadas suas funções biológicas, entre outras análises. Além disso, a quantidade de dados a ser tratada torna impeditiva sua análise sem o uso de computadores.

Dessa maneira, dentro de um projeto genoma as análises computacionais dos dados obtidos por meio dos sequenciadores automáticos são realizadas em diferentes fases. Para cada fase, existe um conjunto de ferramentas de bioinformática a ser utilizado. Porém, cada tipo de pesquisa implica numa combinação diferente de ferramentas, já existentes ou a serem desenvolvidas, de acordo com os objetivos do projeto, e este sistema é chamado de *workflow* de bioinformática. Isso gera uma complexidade adicional ao projeto, pois é necessário gerenciar uma quantidade razoável de ferramentas, assim comos os dados utilizados como suas entradas e saídas.

3.3 Workflows

3.3.1 Conceitos Básicos

Um *workflow* é uma sequência de passos a serem seguidos para atingir um determinado objetivo, podendo ser reproduzido de maneira idêntica em uma segunda execução. Ele é composto de grupos de dados, fases de análise, fluxos e ferramentas ordenados de maneira a se atingir o objetivo desejado [38].

Os *workflows* de bioinformática fazem parte da categoria de *workflows* científicos. Esse grupo caracteriza-se por tentar alcançar um objetivo científico e, normalmente, é expresso em termos de fases de execução e suas dependências, com foco principalmente no fluxo dos dados entre as diferentes fases, ao contrário dos *workflows* de negócio, mais focados nas políticas e regras de negócio [59]. Normalmente, *workflows* são projetados de maneira visual, utilizando ferramentas como diagramas de bloco ou linguagens específicas. Um *workflow* torna-se, assim, uma fonte de conhecimento, como uma "receita" que fornece informação sobre os meios para automatizar, documentar e reproduzir um processo de trabalho.

A execução repetitiva de um grande número de ferramentas interdependentes, além da gestão dos dados utilizados e produzidos durante a execução podem tornar os *workflows* excessivamente complexos e dispendiosos, tornando sua execução manual muito custosa e sujeita a erros.

Para lidar com este problema foram desenvolvidos os Sistemas de Gerenciamento de Workflows (*Workflows Management Systems* - WfMS), cuja função é automatizar a execução de *workflows*. Ademais, esses sistemas também podem oferecer ferramentas de definição e validação de *workflows*, além de fornecer meios de monitoramento em tempo de execução das fases de um *workflow* [59]. Exemplos de sistemas são Kepler [2], Taverna [42], Cyrille2 [30] e Galaxy [38]. Alguns dos requisitos operacionais para tais sistemas são [60]:

- **Alta taxa de processamento**: o sistema deve ser capaz de lidar com grandes conjuntos de dados, complexos *workflows* para análise de dados e grandes quantidades de tarefas que necessitam de longos períodos de processamento.

- **Facilidade de uso**: o sistema deve possuir interfaces gráficas bem projetadas que tornam o *workflow* fácil e intuitivo para usuários leigos.

- **Flexibilidade**: o sistema deve ser flexível o suficiente para que ferramentas novas ou atualizadas sejam incluídas facilmente.
- **Modularidade**: o sistema deve possibilitar ao operador acompanhar mudanças nas bases de dados utilizadas e re-executar somente as partes afetadas do *workflow*, com o mínimo de redundância.
- **Tolerância a falhas**: o sistema deve ser capaz de se recuperar caso recursos falhem, automaticamente reiniciando a fase na qual ocorreu a falha em outro recurso.
- **Reprodutibilidade**: o sistema deve ser capaz de reproduzir as fases do *workflow*, principalmente atento à procedência dos dados utilizados na análise (*data provenance*).

3.3.2 Workflows em Bioinformática

Um *workflow* de bioinformática para projetos de sequenciamento de genomas, geralmente, é formado por um subconjunto das seguintes fases: filtragem, mapeamento, montagem e análise.

A **filtragem** é a fase na qual os arquivos de saída dos sequenciadores automáticos são traduzidos para um formato aceito pelos bancos de dados públicos de SRS produzidas durante os projetos genoma. Esses bancos armazenam e publicam os dados para que outros cientistas tenham acesso, e para registrar experimentos realizados. O exemplo mais conhecido é o banco de dados do *National Center for Biotechnology Information* (NCBI) [66] dos Estados Unidos. Lá estão armazenados dados sobre DNA, RNA, genes, proteínas, entre outros. O banco de dados do NCBI recebe os dados produzidos pelos sequenciadores automáticos em diversos formatos, dentre eles: *Sequence Read Archive* (SRA), *Standard Flowgram Format* (SFF) e FASTQ [32]. O *software* utilizado nesta fase é, normalmente, fornecido pelo fabricante do sequenciador. Mas, como alguns dos formatos são padronizados, é possível a implementação de *software* de filtragem por terceiros. Um exemplo é a ferramenta Flower [61].

O **mapeamento** é a fase na qual as SRS produzidas pelos sequenciadores automáticos, já em formato padronizado após a fase de filtragem, são mapeadas a um genoma de referência. Esta fase é necessária quando são utilizados os sequenciadores automáticos de alto desempenho. Isso se dá pois as SRS produzidas por estes sequenciadores são geralmente menores que as produzidas no sequenciamento tradicional, o que dificulta a montagem dos fragmentos na sequência original. Para resolver este problema, as SRS são mapeadas ao genoma de um indivíduo diferente do mesmo organismo ou ao genoma de um organismo semelhante. Assim, com várias SRS mapeadas em uma mesma região do genoma de referência é possível realizar a montagem da sequência original utilizando técnicas tradicionais. Além disso, a saída do próprio mapeamento pode ser utilizada diretamente por várias outras análises. Exemplos de ferramentas tipicamente usadas nesta fase são Bowtie [52], BWA [54] e RMAP [83].

A **montagem** é a fase na qual as SRS produzidas pelos sequenciadores automáticos são montadas de forma a reproduzir a sequência genética original. Para as SRS produzidas por sequenciadores de alto desempenho, normalmente, são utilizadas duas técnicas. A primeira é mapear as SRS a um genoma de referência, e depois aplicar a montagem

utilizando as técnicas tradicionais. Outra maneira é a chamada montagem *de novo*, por meio da qual é possível montar a sequência original a partir somente das SRS, sem a necessidade do uso de um genoma de referência. Esta montagem pode ser feita por meio de extensões realizadas nas SRS originais, utilizando tabelas *hash* para mapear SRS de um mesmo trecho da sequência original. Outra técnica seria a aplicação dos chamados grafos *de Bruijn*, os quais representam sobreposições de SRS. A saída dessa fase são arquivos contendo *contigs* (um grupo de dois ou mais fragmentos representando um sequência obtida), *singlets* (fragmentos que não puderam ser agrupados) e outros dados auxiliares. Ferramentas para montagem de SRS produzidas por sequenciadores de alto desempenho comumente utilizadas são Velvet [97], SOAPdenovo [56] e ABySS [10].

Finalmente, na fase de **análise** é feito o tratamento das informações obtidas nas fases anteriores, o qual depende do objetivo do projeto sendo executado. A análise mais comum realizada nesta fase é a anotação, a qual tem como alvo a descoberta das funções biológicas de *contigs* e *singlets* produzidas na fase de montagem, ou de regiões dos genomas de referência que tiveram SRS mapeadas durante a fase de mapeamento. Para isso, são usados algoritmos de comparação aproximada de sequências, além de informações armazenadas em bancos de dados. A ferramenta clássica para executar a anotação é chamada BLAST [3]. Outros tipos de análises também podem ser feitos, com diferentes objetivos, como descobrir novos genes, identificar RNAs não-codificadores, identificar genes diferencialmente expressos, entre outros. Algumas das ferramentas deste grupo são SOAPsnp [55], TopHat [90], PORTRAIT [6] e HMMER [26].

Como pode ser visto, a depender do objetivo do projeto em execução, um *workflow* específico deve ser projetado, com fases, fluxo de dados e ferramentas também específicos. Um primeiro exemplo de *workflow* é formado pelas fases de filtragem, mapeamento, montagem e análise. Neste caso, durante a análise é realizada a anotação. A Figura 3.2 descreve graficamente este *workflow*.

Outro possível *workflow* é constituído pelas fases de filtragem, montagem e análise, como a anotação. Neste caso, as SRS disponíveis no banco de dados após a filtragem devem ter tamanho suficiente para aplicar uma montagem *de novo*. A Figura 3.3 mostra esquematicamente este *workflow*.

Por fim, há ainda um terceiro possível *workflow*, formado pelas fases de filtragem, mapeamento e análise, esta podendo ser, por exemplo, a detecção do chamado *single nucleotide polymorphism* (SNP), o qual pode indicar uma possível mutação genética entre indivíduos da mesma espécie, entre outras coisas. A Figura 3.4 mostra o *workflow* citado.

3.4 Nuvens Computacionais e Bioinformática

A computação em nuvem tem sido utilizada como alternativa para tratar a grande quantidade de dados produzidos pelos projetos genoma. Alguns motivos são: capacidade de oferecer uma infraestrutura computacional flexível e sob demanda; recursos aparentemente ilimitados e que seguem um modelo *pay-per-use*; e possibilidade de distribuição de processamento em larga escala.

Essas características permitem que pesquisadores implementem ferramentas de bioinformática que atingem um alto grau de paralelismo de maneira simplificada, acarretando em redução de tempo de execução, sem aumentar a complexidade no seu desenvolvimento.

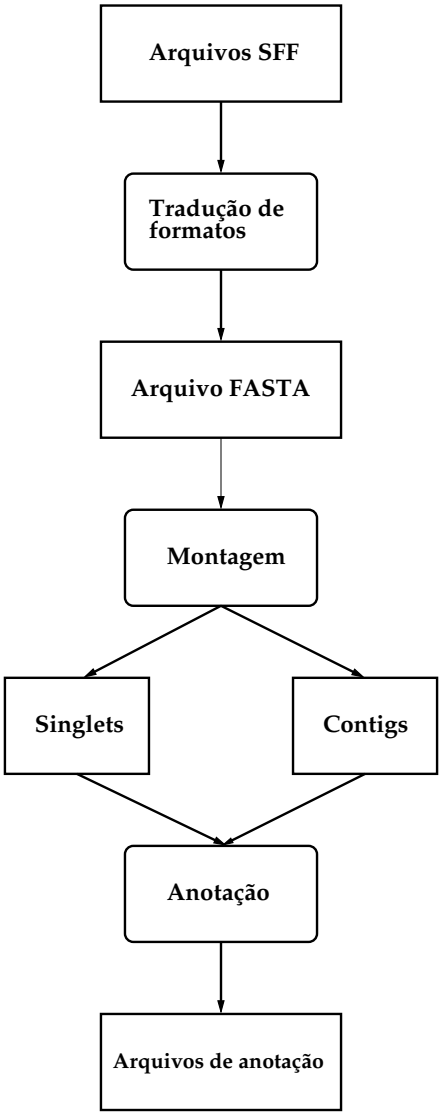

Figura 3.2: Exemplo de *workflow* para projetos de sequenciamento genômico para máquinas 454.

Figura 3.3: Exemplo de *workflow* com utilização de montagem *de novo*.

Figura 3.4: Exemplo de *workflow* com análise feita logo após o mapeamento das SRS.

Uma das tecnologias mais utilizadas para a implementação de ferramentas de bioinformática para execução em ambiente de nuvem computacional é o *framework* **Apache**

Hadoop [36], do qual a implementação do modelo *MapReduce* [23] e o seu sistema de arquivos distribuído HDFS são utilizados como infraestrutura para a distribuição de processamento e armazenamento de dados em larga escala.

Uma forma de utilizar o *framework* é construir algoritmos de bioinformática voltados para o modelo *MapReduce*. A ferramenta *CloudBurst* [82] é a implementação de um algoritmo paralelo projetado para o modelo *MapReduce* com o objetivo de mapear as SRS a um genoma de referência. Seu tempo de execução varia linearmente com o número de SRS mapeadas, e quase linearmente com o aumento de processadores utilizados. Ao fazer o mapemanento de milhões de SRS ao genoma humano, a aplicação chega a ser trinta vezes mais rápida se comparada com outras aplicações não distribuídas [52, 83].

A ferramenta *CloudBurst* utiliza os bem conhecidos algoritmos de *send-and-extend* para realizar o mapeamento de SRS para um genoma de referência. Na fase de *map* do modelo *MapReduce*, o algoritmo da ferramenta encontra possíveis alinhamentos exatos, as *seeds*, entre as SRS e os genomas. Em seguida, o modelo prevê uma fase chamada *shuffle*, quando os alinhamentos são agrupados por *seed*, isto é, trechos do genoma com a mesma *seed* são agrupados com SRS que também possuem a mesma *seed*. Finalmente, na fase de *reduce*, as *seeds* das SRS são estendidas realizando comparações com o restante do trecho do genoma de referência que foi alinhado na fase de *map*. O poder da ferramenta se encontra no fato de que o mapeamento, o agrupamento e a extensão das *seeds* são feitas de maneira paralela em centenas de processadores. A Figura 3.5 descreve graficamente o algoritmo utilizado pela *CloudBurst*.

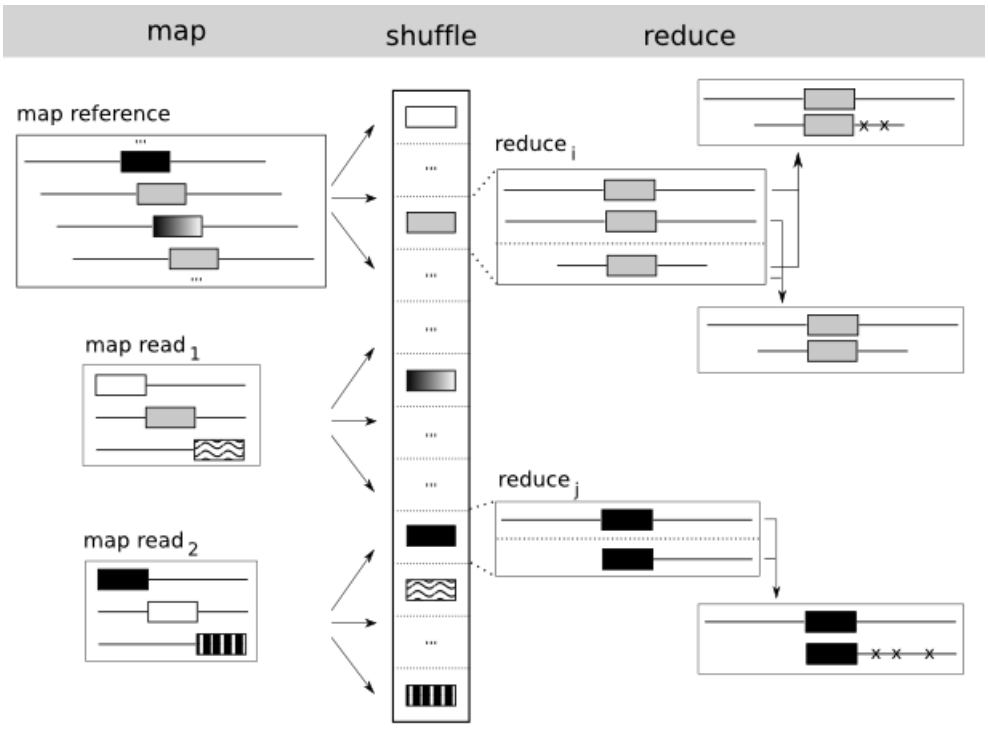

Figura 3.5: Funcionamento do algoritmo *CloudBurst* [82].

Uma outra forma de utilizar o *framework* como infraestrutura é construir *workflows* que sejam formados pelas duas fases do modelo: *map* e *reduce*. A construção é feita por meio do modo *Streaming*, disponível no *framework*. A ferramenta *Crossbow* [51] é

um *workflow* implementado desta maneira, no qual o Bowtie [51], uma ferramenta de mapeamento de SRS, executa durante a fase de *map*, sendo sua saída processada pelo SOAPsnp [55], uma ferramenta que identifica SNPs, e executada durante a fase de *reduce*. Usando a ferramenta *Crossbow*, o tempo de execução do reconhecimento de SNPs entre um conjunto de aproximadamente 2,6 bilhões de SRS e todo o genoma humano como referência foi de um pouco mais de 3 horas em um *cluster* de 320 núcleos montado na infraestrutura da Amazon EC2 [57]. O custo da execução do experimento foi de menos de 100 dólares. Durante a execução do *workflow*, as SRS são enviadas como entrada para os nós do *cluster* Hadoop que executarão a fase de *map*. Nesta fase, as SRS são mapeadas ao genoma de referência utilizando a ferramenta Bowtie. Na sequência, os mapeamentos são agrupados por trecho do genoma de referência, e cada grupo é enviado para um nó que realiza a fase de *reduce*. Nela, a ferramenta SOAPsnp é utilizada para a detecção de SNPs no trecho do genoma sendo analisado. A Figura 3.6 demonstra o funcionamento da ferramenta.

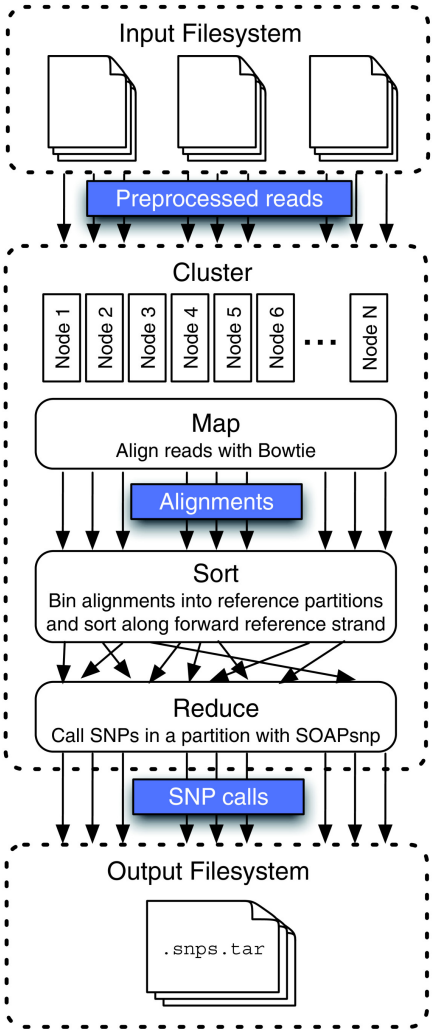

Figura 3.6: Funcionamento do *pipeline Crossbow* [51].

Um outro exemplo de *workflow* de bioinformática desenvolvido com a tecnologia da

computação em nuvem é a ferramenta *Myrna* [50]. Ela é utilizada para identificar genes diferencialmente expressos em conjuntos grandes de dados sequenciados. O *workflow* combina uma fase de mapeamento com uma de análise estatística, realizada pela ferramenta R [72], que é capaz de analisar mais de 1 bilhão de SRS em um pouco mais de uma hora e meia, utilizando 320 núcleos ao custo de aproximadamente 75 dólares.

Pratt *et al.* [69] adaptaram um motor de busca de peptídeos chamado X!Tandem também para o Hadoop MapReduce. A aplicação resultante, MR-Tandem, executa em qualquer *cluster* Hadoop, mas foi projetada especialmente para o Amazon EC2. Para isso, os pesquisadores modificaram o código C++ do X!Tandem e criaram um *script* Python para executá-lo em *clusters* Hadoop, por meio também do modo *Streaming*.

A computação em nuvem também foi utilizada recentemente na área de genômica comparativa. O algoritmo RSD (*Reciprocal Smallest Distance*), uma composição de diversas ferramentas de bioinformática, foi adaptado para ser executado na infraestrutura da Amazon EC2, obtendo resultados expressivos [93].

Zhang *et al.* [98] utilizaram a computação em nuvem como ferramenta para análise de conjuntos de genes. Eles desenvolveram um algoritmo para identificação de biomarcadores em conjuntos de genes para ser executado em nuvem. A ferramenta, chamada de YunBe, está pronta para ser executada na infraestrutura da Amazon. Ela obteve um bom desempenho em comparação com execuções em *desktops* e *clusters*.

Ekanayake *et al.* [27] portaram duas aplicações de bioinformática — uma de alinhamento de duas sequências Alu e outra para montagem de sequências expressas (EST) — para as tecnologias de computação em nuvem Apache Hadoop e Microsoft DryadLINQ. Eles estudaram o desempenho das duas aplicações nos dois ambientes, comparando com a implementação tradicional para *clusters*, que utilizava MPI. Eles também analisaram como dados não homogêneos afetavam os mecanismos de escalonamento das infraestruturas de nuvem, comparando seu desempenho em *hardware* real e virtualizado.

Seguindo o exemplo acima, recentemente outras aplicações de bioinformática foram portadas para nuvens computacionais [41, 46, 48], e algumas de suas notórias características são a facilidade de uso, por meio de interfaces *web*, e eficiência na execução de ferramentas que fazem uso intensivo de memória e armazenamento.

Embora utilizem o poder computacional oferecido pela computação em nuvem, as ferramentas acima descritas são aplicadas para problemas ou análises específicas, não pretendendo ser uma solução completa e flexível para a aplicabilidade de *workflows* complexos de bioinformática em nuvem. Nessa direção, alguns esforços tem sido realizados para oferecer uma arquitetura de simples utilização por parte de pesquisadores que desejam combinar e executar diferentes aplicações em *workflows* de bioinformática.

Um exemplo é a integração do Hadoop com a ferramenta de gerenciamento de *workflows* Kepler [2, 94]. Por meio desta integração é possível concatenar aplicações MapReduce com outros tipos de tarefas. No entanto, essa ferramenta oferece somente uma implementação genérica de um tarefa MapReduce, tendo o pesquisador que customizá-la de acordo com suas necessidades, tornando o processo de criação de *workflows* mais complicado.

Com o intuito de oferecer uma solução mais completa, Angiuoli *et al.* [4] apresentaram a ferramenta CloVR. Ela é uma máquina virtual que pode ser executada em computadores pessoais ou em infraestruturas de computação em nuvem, e também pode ser utilizada de maneira integrada, usando os recursos de provedores de infraestrutura gerenciados au-

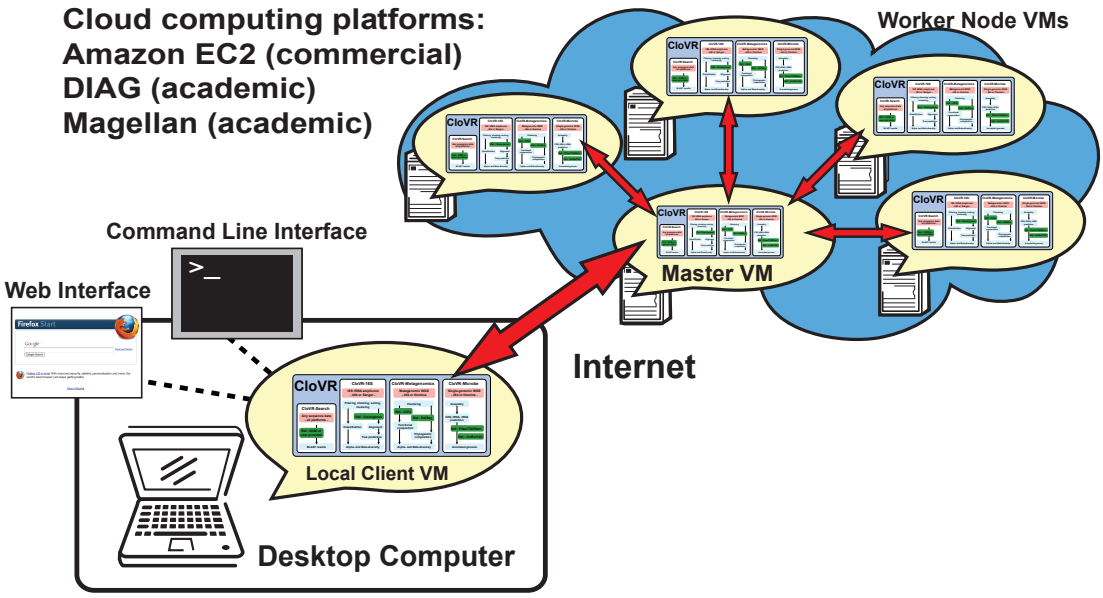

Figura 3.7: Arquitetura da ferramenta CloVR [4].

tomaticamente pela máquina virtual executando locamente. A máquina virtual em si é um sistema operacional completo, e nele já estão instaladas diversas ferramentas de bioinformática para executar os *workflows* pré-definidos. Além disso, em ambientes de computação em nuvem a ferramenta CloVR é capaz de criar novas instâncias de si mesma dinamicamente, de acordo com o tamanho do conjunto de dados sendo analisado, como mostrado na Figura 3.7, que descreve graficamente sua arquitetura. Assim, embora seja uma ferramenta muito versátil, ela não permite ao pesquisador customizar *workflows*, estando ele restrito aos *workflows* pré-definidos. Ademais, no que se refere ao ambiente de computação em nuvem existem algumas limitações. Por exemplo, ao utilizar diferentes infraestruturas ela não faz uso das mesmas de maneira direta, ou seja, das aplicações e recursos que porventura já estejam disponíveis em uma determinada instituição. Ao contrário, ela necessita que seja possível iniciar uma instância de si mesma na infraestrutura, caso contrário a integração não é possível. Por fim, a intenção da ferramenta, pelo menos em princípio, é de ser uma aplicação para execução de *workflows* que é capaz de utilizar recursos das infraestruturas de terceiros, mas não exatamente construir uma federação com os requisitos que isso acarreta, como indicado na Seção 2.2.

De uma forma mais centralizada, a plataforma Bio-Cloud [8] oferece aplicações de bioinformática como serviço de forma transparente, sem detalhar como sua infraestrutura é utilizada. Ela foi projetada para suportar processamento de larga escala, integrando diferentes centros computacionais, com desempenho de até 157 Tflops, com 33.3TB de memória e 12.6 PB de armazenamento.

Capítulo 4

Arquitetura BioNimbus

Neste trabalho é proposta a arquitetura BioNimbus [79] para federação de nuvens computacionais híbrida, a qual permite a integração e o controle de diferentes provedores de infraestrutura, com suas ferramentas de bioinformática oferecidas como serviço, de maneira transparente, flexível e tolerante a falhas, com grande capacidade de processamento e armazenamento. Neste contexto, procura-se oferecer a ilusão de que os recursos computacionais disponíveis são ilimitados e que as demandas dos usuários sempre serão atendidas.

Para atingir esses objetivos, a arquitetura BioNimbus permite a integração entre diferentes plataformas de computação em nuvem, o que significa que provedores independentes, heterogêneos, privados ou públicos de nuvens computacionais podem oferecer seus serviços de bioinformática conjuntamente, mantendo suas características e políticas internas.

Na Seção 4.1 é apresentada uma visão gera da arquitetura proposta e na Seção 4.2 cada componente da arquitetura é descrito mais detalhadamente. Em seguida, na Seção 4.3 são apresentados os casos de uso previstos para a arquitetura, com um detalhamento da interação entre seus componentes para que cada caso de uso seja atendido. Por fim, na Seção 4.4 é apresentada uma discussão sobre como a arquitetura BioNimbus atende os requisitos de uma federação de nuvens, além de uma breve comparação com outras arquiteturas propostas.

4.1 Visão Geral

Todos os componentes da arquitetura BioNimbus e suas respectivas funcionalidades estão bem definidos e divididos, o que permite simplicidade e eficiência quando novos provedores de nuvens desejarem ser incluídos na federação criada, como mostra a Figura 4.1. Outra característica importante é que a comunicação entre esses componentes é realizada por meio de uma rede *Peer-to-Peer* (P2P) [85].

Para que seja possível a integração de um provedor de nuvem computacional na federação, um serviço ***plug-in*** `de integração` é utilizado para realizar a comunicação entre o provedor e os serviços controladores da federação, mapeando as requisições feitas no padrão da federação para o formato de requisição específico daquele provedor. Além disso, o *plug-in* é responsável por coletar informações sobre os recursos computacionais disponíveis e quais ferramentas de bioinformática são oferecidas para utilização por parte

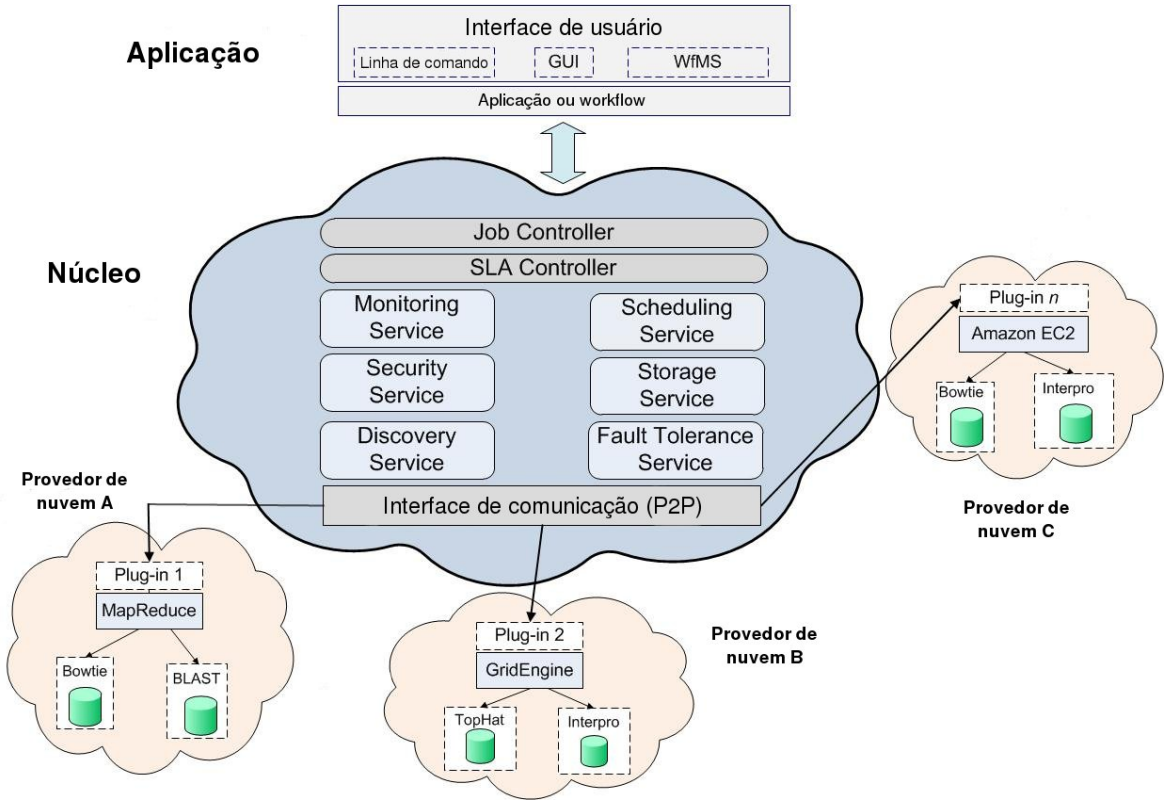

Figura 4.1: BioNimbus: uma arquitetura de federação de nuvens computacionais para aplicações de bioinformática.

do usuários da federação. A utilização desses *plug-ins* torna a arquitetura flexível para a integração dos mais diferentes tipos de infraestruturas de nuvem computacional.

Os serviços controladores citados acima fazem parte do núcleo da arquitetura BioNimbus, os quais são:

- ***Discovery Service***: coleta dados sobre as diferentes nuvens computacionais federadas pela arquitetura, como recursos disponíveis e ferramentas oferecidas como serviço;
- ***Scheduling Service***: escalona, inicializa, acompanha e finaliza execuções das ferramentas de bioinformática;
- ***Storage Service***: coordena a estratégia de armazenamento de arquivos consumidos e produzidos pelas ferramentas executadas;
- ***Fault Tolerance Service***: acompanha os membros da rede P2P onde executam esses serviços controladores, bem como aqueles que coordenam as infraestruturas federadas, de maneira que seja possível recuperar os serviços ou requisições de execução de ferramentas em caso de falha.
- ***Monitoring Service***: registra, monitora e armazena dados sobre carga das nuvens da federação e sobre histórico de execução de aplicações;

- ***Security Service***: gerencia a política de acesso à federação e as credenciais de acesso dos usuários a cada uma das nuvens.

A integração desses serviços na arquitetura é feita por meio de uma interface que pode ser usada para adicionar novas e não previstas funcionalidades à arquitetura, sem impactar em seu funcionamento, e aumentando sua flexibilidade. Esta interface exige que sejam implementadas funções operacionais do serviço, como inicialização, atualização de estado e finalização do serviço, bem como oferece a sinalização de eventos na rede P2P.

Por último, estão as funcionalidades de interação com os usuários. Para executar aplicações ou montar *workflows* de bioinformática, a arquitetura prevê que existam interfaces *web*, linhas de comando ou outras formas mais complexas de interação, como sistemas gerenciadores de *workflows*, ou *WfMS* [38, 42].

Toda a comunicação entre os componentes da arquitetura, ou seja, serviços *plug-in*, serviços controladores e aplicações de interação com o usuário, é realizada por meio de uma rede P2P. Cada componente envia suas requisições como pares de mensagens de requisição (*request*) e resposta (*reply*), definidas para cada tipo de requisição prevista pela arquitetura.

Além dos componentes indicados acima, a arquitetura também define algumas classes de informação que são trocadas entre eles, para que sirvam de referência para as implementações. As classes de informação são:

- `PeerInfo`: informações referentes a um *peer*. Um *peer* é um membro da rede P2P. Exemplo: identificador único, endereço de rede, latência de rede e tempo na rede;

- `PluginInfo`: informações referentes a um *plug-in* de integração, já definido nesta seção. Exemplo: dados da classe `PeerInfo` que descrevem o *peer* onde o *plug-in* executa, número total de CPUs, número de CPUs livres, tamanho total de armazenamento de dados e tamanho de armazenamento livre;

- `ServiceInfo`: informações referentes a um serviço oferecido por um provedor de nuvem computacional que faz parte da federação. Um **serviço** para a arquitetura BioNimbus é uma ferramenta de bioinformática. Exemplo: identificador único, nome da ferramenta, parâmetros de configuração, arquivos de entrada e arquivos de saída;

- `JobInfo`: informações referentes a um *job*. Um ***job*** é uma requisição de execução de um serviço na federação. Exemplo: identificador único, identificador do serviço solicitado e argumentos para o serviço;

- `TaskInfo`: informações referentes a uma *task*. Uma ***task*** é uma instância de um *job*. Representa uma execução de um *job* em algum *plug-in* de integração. Essa diferenciação entre *job* e *task* é importante para que seja possível, por exemplo, haver execuções paralelas de um mesmo *job* em infraestruturas diferentes da federação. Exemplo: identificador único, as informações da classe `JobInfo` correspondente, a *PluginInfo* do local de execução e seu estado de execução na infraestrutura, os quais podem ser "EM ESPERA", "EM EXECUÇÃO" e "FINALIZADO"

- `FileInfo`: informações referentes a um arquivo armazenado na federação. Exemplo: identificador único, nome e tamanho do arquivo;

- `PluginFileInfo`: informações referentes a uma instância de um arquivo. Um arquivo pode conter mais de uma instância ser for implementada a replicação de arquivos em diferentes infraestruturas da federação. Exemplo: identificador único, sua `FileInfo` e a *PluginInfo* do local de armazenamento.

A seguir, cada grupo de componentes, com suas funcionalidades e mensagens definidas pela arquitetura BioNimbus, é descrito mais detalhadamente.

4.2 Descrição dos Componentes

4.2.1 *Plug-in* de Integração das Nuvens Computacionais

Os *plug-in* de integração tem como objetivo ser a interface de comunicação entre o seu respectivo provedor de serviço e os demais componentes da arquitetura BioNimbus, e também entre ele e os demais provedores federados. Para que a comunicação seja feita com sucesso, o *plug-in* precisa mapear as requisições vindas dos componentes da arquitetura para ações correspondentes a serem realizadas na infraestrutura do provedor de serviço. Por isso, para cada tipo de infraestrutura é preciso haver uma implementação diferente do *plug-in*.

O *plug-in* necessita tratar três tipos de requisição: informações sobre a infraestrutura do provedor de serviço, gerenciamento de tarefas e transferências de arquivos. Para informações sobre o provedor, existe um par de mensagens:

- `InfoReq`: requisição de informações sobre a a infraestrutura do provedor para o *plug-in* de integração. Ela não possui nenhum conteúdo adicional;
- `InfoReply`: resposta do *plug-in* de integração com informações sobre a infraestrutura do provedor, ou seja, da classe `PluginInfo`.

Para as requisições de gerenciamento de tarefas, existem três pares de mensagens. O primeiro refere-se ao início de uma *task* na infraestrutura do provedor:

- `TaskStartReq`: requisição de início da execução (*task*) de um serviço na infraestrutura do provedor. O conteúdo da mensagem é constituído de informações sobre o *job* do usuário, ou seja, da classe `JobInfo`;
- `TaskStartReply`: resposta do *plug-in* de integração enviada após o início da execução da *task* na infraestrutura do provedor. Seu conteúdo é formado por dados sobre a *task*, ou seja, da classe `TaskInfo`, mais os dados sobre o *job* (`JobInfo`) que estavam contidos na requisição.

O segundo par de mensagens se refere a consultas de estado de execução de *tasks* executando nas infraestruturas dos provedores:

- `TaskStatusReq`: requisição de informações sobre o estado de execução de uma *task*. Deve conter, pelo menos, seu identificador único;
- `TaskStatusReply`: resposta do *plug-in* de integração sobre o estado atual da *task*, ou seja, dados da classe `TaskInfo`.

O último par de mensagens sobre gerenciamento de tarefas refere-se ao cancelamento da execução de *tasks* na infraestrutura do provedor:

- `TaskCancelReq`: requisição de cancelamento de execução de uma *task*. Deve conter, pelo menos, seu identificador único;
- `TaskCancelReply`: resposta do *plug-in* de integração confirmando o cancelamento da execução. Contém dados da classe `TaskInfo` sobre a *task* cancelada.

Adicionalmente às mensagens detalhadas acima, referentes ao gerenciamento de tarefas, o *plug-in* de integração deve enviar uma mensagem do tipo `TaskEnd` para o *Scheduling Service* para que este atualize o estado de execução desta tarefa para o estado de finalizada. Essa mensagem será detalhada quando o *Scheduling Service* for descrito na Seção 4.2.2.

Finalmente, para as requisições de transferências de arquivos existem também três pares de mensagens. O primeiro deles refere-se à preparação que deve haver na infraestrutura do provedor antes que seja possível ao usuário efetivamente baixar um arquivo. Essa preparação pode ser necessária caso, por exemplo, o arquivo esteja armazenado em um sistema de arquivos ou banco de dados não acessível diretamente no momento do *download*. As mensagens são:

- `FilePrepReq`: requisição de preparação para futuro *download* de um arquivo localizado na infraestrutura do provedor. Deve conter dados da classe `PluginFileInfo`;
- `FilePrepReply`: resposta do *plug-in* de integração sinalizando o fim da preparação para o futuro *download* indicado pela requisição. Deve conter dados da classe `PluginFileInfo`.

O segundo par de mensagens para transferências de arquivos é usado quando se deseja realizar efetivamente o *download* de um arquivo:

- `FileGetReq`: requisição de *download* de um arquivo localizado na infraestrutura de um provedor. Deve conter dados da classe `PluginFileInfo`;
- `FileGetReply`: resposta do *plug-in* de integração sinalizando o fim do envio de arquivo para quem fez a requisição. Deve conter pelo menos o identificador do arquivo.

Por último, é descrito o par de mensagens para realizar um *upload* de arquivo na federação:

- `FilePutReq`: requisição de *upload* de um arquivo para a infraestrutura de um provedor. Deve conter dados da classe `FileInfo` e o próprio conteúdo do arquivo enviado;
- `FilePutReply`: resposta do *plug-in* de integração enviada ao *peer* que realizou o *upload*, enviada após a gravação efetiva do arquivo na infraestrutura do provedor, que pode ser um banco de dados, um sistema de arquivos distribuído, etc.

Adicionalmente às mensagens detalhadas acima, referentes à transferência de arquivos, o *plug-in* de integração deve enviar uma mensagem do tipo `StoreAck` para o *Storage Service* para que este atualize sua tabela de arquivos. Essa mensagem será detalhada quando o *Storage Service* for descrito na Seção 4.2.2.

4.2.2 Serviços Controladores da Federação (Núcleo)

O chamado **núcleo** da arquitetura BioNimbus é responsável por gerenciar a federação de nuvens computacionais. Entre suas funções estão o descobrimento de provedores e recursos, o escalonamento e o acompanhamento das execuções de tarefas, e a estratégia de armazenamento de arquivos. Para cada possível função de gerenciamento necessária, um serviço controlador pode ser adicionado à arquitetura.

Inicialmente, são propostos seis serviços principais: *Discovery Service*, *Security Service*, *Monitoring Service*, *Scheduling Service*, *Storage Service* e *Fault Tolerance Service*. Além disso, são propostos dois controladores: *Job Controller* e *SLA Controller*. Para exercer suas funcionalidades, os serviços e os controladores também interagem com os demais componentes da arquitetura por meio de mensagens enviadas pela rede P2P.

Abaixo, segue uma descrição mais detalhada de quais são as funções de cada serviço controlador proposto, e quais tipos de mensagens são usados para a interação com os demais componentes da arquitetura.

Discovery Service

O ***Discovery Service*** é responsável por identificar os provedores de serviço que fazem parte da federação de nuvens computacionais, e consolidar as informações sobre capacidade de armazenamento e processamento, assim como quais ferramentas de bioinformática estão sendo oferecidas pelo provedor como serviço, com detalhes sobre parâmetros de execução e arquivos de entrada e saída.

O *Discovery Service*, para realizar sua função, primeiramente envia mensagens do tipo `InfoReq` por meio da rede P2P. Essas mensagens, como descrito na Subseção 4.2.1, são tratadas pelos serviços *plug-ins* dos provedores. As respostas, mensagens do tipo `InfoReply`, contêm todos os dados referentes à infraestrutura e às ferramentas de bioinformática disponíveis.

Com relação à entrada de um provedor na federação construída pela arquitetura BioNimbus, *a priori*, qualquer integrante (*peer*) da rede P2P pode responder às mensagens `InfoReq` enviadas pelo *Discovery Service*, oferecendo recursos computacionais e aplicações de bioinformática como serviço. Entretanto, por questões de controle e de segurança, a permissão para fazer parte da federação como provedor de serviços deve ser verificada com a ajuda do *Security Service* sempre que qualquer informação sobre um novo provedor chegar ao *Discovery Service*.

Para consolidar os dados sobre as nuvens federadas, o *Discovery Service* deve manter uma estrutura de dados e, para cada nova resposta vinda dos *plug-ins* dos servidores, estes dados são incrementados ou atualizados. Além disso, o serviço deve ter uma política de expiração de provedores, removendo de sua estrutura de dados aqueles que não façam mais parte da federação.

Finalmente, tendo realizado a consolidação das informações sobre o estado atual da federação de nuvens, o *Discovery Service* é capaz de atender às requisições enviadas pelos demais componentes da arquitetura que são de sua responsabilidade. Essas requisições formam um par de mensagens, as quais são:

- `CloudReq`: requisição de informações sobre o estado atual da federação de nuvens. Ela não possui nenhum conteúdo adicional;

- `CloudReply`: resposta do *Discovery Service* com uma coleção de dados da classe `PluginInfo` que representa a consolidação dos dados coletados pelo serviço até o momento, indicando os provedores integrantes da federação.

Com a descrição acima, é possível perceber que um mecanismo de descoberta de recursos eficiente exerce um importante papel em uma federação de nuvens, pois a informação coletada por esse serviço é essencial para o funcionamento adequado de todos os demais serviços. De acordo com Oppenheimer *et al.* [67], em sistemas distribuídos de larga escala federados, uma infraestrutura de descoberta de recursos deve atender os seguintes requisitos chave:

- Ser escalável ao ponto de conseguir lidar com milhares de máquinas sem ficar indisponível ou perder desempenho;
- Ser capaz de lidar tanto com recursos estáticos quanto dinâmicos; e
- Ser flexível o suficiente de forma que requisições possam ser estendidas com o intuito de lidar com diferentes tipos de recursos.

Possíveis implementações de um serviço de descoberta de recursos poderiam usar abordagens centralizadas ou hierárquicas. Porém, elas são conhecidas por possuírem sérias limitações de escalabilidade, tolerância a falhas e sobrecarga de rede [74]. Para a arquitetura BioNimbus, recomenda-se nas implementações que o *Discovery Service* seja descentralizado, e que utilize uma estrutura de dados distribuída, como uma *Distributed Hash Table* (DHT) [7], com o intuito de alcançar baixos custos de gerenciamento da informação e de sobrecarga de rede, busca eficiente de recursos e alta tolerância a falhas. Para o tratamento das informações sobre recursos, aconselha-se o uso de formatos serializáveis e extensíveis como o formato JSON [22]. Desta maneira, será possível para a federação lidar com diferentes tipos de informações com o menor impacto possível.

Job Controller

O *Job Controller* faz a ligação entre o núcleo da arquitetura e a camada de aplicações de interação com o usuário. Uma de suas atribuições é realizar o controle de acesso dos usuários que tentam acessar a federação para realizar pedidos de execução. Para fazer a verificação de credenciais, o controlador acessa o *Security Service*. Além disso, o *Job Controller* é responsável por gerenciar os pedidos dos vários usuários, de forma a fazer o controle por usuário, e manter os resultados para posterior consulta.

Security Service

Uma federação de nuvens computacionais, ao implementar um serviço de segurança, precisa considerar as políticas de cada provedor de serviço individualmente sem, contudo, criar uma interdependência entre elas. Este é um dos principais requisitos do *Security Service*. Além disso, ele deve criar contextos de segurança entre usuários e provedores de serviço, e estabelecer políticas de controle de acesso para que um membro da rede P2P faça parte da federação de nuvens como provedor.

A criação de um contexto de segurança possui requisitos divididos em três tópicos principais: autenticação, autorização e confidencialidade. Para a arquitetura BioNimbus, é proposta a utilização de protocolos e algoritmos padrão, muito utilizados em ambientes de computação em nuvem.

Devido à natureza descentralizada de uma federação de nuvens computacionais, a **autenticação** não deve seguir uma abordagem centralizada, pois impõe limites para a escalabilidade da federação e cria interdependência entre provedores. Uma alternativa mais robusta para esse ambiente é a utilização de um protocolo *Single Sign-On* (SS0) [68] de forma que não seja necessária uma autoridade central responsável pela autenticação de usuários. Esta técnica evita pontos únicos de falha e permite maior escalabilidade no número de usuários. Um exemplo de mecanismo SSO é o padrão OpenID [75]. Ele tem sido extensivamente usado por empresas e instituições acadêmicas pelo mundo, principalmente no ambiente de computação em nuvem. O OpenID permite que cada sítio (exemplo, uma nuvem) forneça um mecanismo de autenticação para seus usuários e haja como um provedor de credenciais. Uma vez que o usuário está autenticado em um provedor, suas credenciais podem ser validadas por meio do protocolo OAuth [9] por outros provedores, sem expor dados sobre a conta do usuário e sem a necessidade de uma autenticação específica para cada provedor.

No que se refere à **autorização**, ela é implementada por meio de listas de controle de acesso (ACL) [89] fornecidas por cada provedor de serviço. Uma ACL especifica quem pode ter acesso a um determinado recurso, como um diretório de um sistema de arquivo ou uma aplicação de bioinformática. Portanto, cada provedor é capaz de determinar padrões de acesso, mantendo o controle de seus recursos sem criar interdepedência entre provedores.

Por fim, para atender o requisito de **confidencialidade**, o sigilo deve estar principalmente na troca de informações entre os membros da federação, os componentes controladores da arquitetura e os usuários. Ele pode ser implementado por meio do uso de conexões SSL/TLS [87], por exemplo. A arquitetura BioNimbus não obriga a utilização de conexões criptografadas, mas elas podem ser utilizadas sem impacto algum. Basta que cada membro da federação forneça certificados para a negociação dos canais seguros.

SLA Controller

De acordo com Wu & Buyya [95], um *Service-level Agreement* (SLA) é um contrato formal entre provedores de serviço e consumidores para garantir que as expectativas de qualidade de serviço do usuário sejam atingidas. Na arquitetura BioNimbus, o *SLA Controller* é responsável por implementar o chamado ciclo de vida de SLA, o qual possui seis passos: descoberta de provedores de serviço, definição de SLA, estabelecimento do acordo, monitoramento de violação do acordo, terminação de acordo e aplicação de penalidades por violação.

Para identificar parâmetros de acordo, são usados os chamados *templates*. Um *template* de SLA representa, entre outras coisas, os parâmetros de QoS (*Quality-of-Service*) que um usuário negociou com a arquitetura. Ele preenche esse *template* por meio da camada de interação com o usuário com os valores necessários, os quais podem descrever requisitos funcionais — como número núcleos de CPU, frequência de CPU, tamanho de memória, versão mínima de aplicações ou tamanho de armazenamento — e não funcionais — como

tempo de resposta, custo a pagar, taxa de transferência de arquivos, disponibilidade ou tempo máximo de execução.

O *SLA Controller* tem a responsabilidade de investigar se os requisitos especificados no *template* preenchido pelo usuário podem ser suportados pela federação de nuvens naquele dado momento. Para tomar essa decisão, o controlador utiliza os dados de SLA informados pelo *plug-in* de integração de cada provedor. Para cada pedido de execução feito pelo usuário para a arquitetura BioNimbus, a negociação de SLA procede da seguinte forma:

1. O usuário faz seu pedido com um *template* preenchido com os parâmetros de SLA desejados à arquitetura;
2. O *SLA Controller* cria uma sessão de negociação, se ela ainda não existe, e repassa o pedido de execução para o *Monitoring Service* com os requisitos mínimos extraídos do *template*;
3. O serviço requisita então um escalonamento para o *Scheduling Service* dado aqueles parâmetros e aguarda sucesso ou falha;
4. Em caso de falha, o processo é reiniciado com um novo *template* SLA até que se chegue a um acordo.

Após a negociação de um acordo ter obtido sucesso, o *SLA Controller* mantém a sessão do acordo por meio de um identificador único, retornando-o para o *Job Controller*, que inclui essa informação na sessão do usuário. Com o pedido do usuário em execução, é preciso acompanhá-lo de forma que o acordo não seja violado. Essa é uma das responsabilidade do *Monitoring Service*.

Monitoring Service

Este serviço recebe pedidos de execução de aplicações (*jobs*) vindos do *Job Controller*, verifica se a aplicação requisitada está disponível em algum provedor de serviço e envia o pedido para o *Scheduling Service*. Após o envio, o serviço monitora a federação para garantir que todos os pedidos foram devidamente atendidos e executados. Ao fim da execução de um *job*, o serviço informa o *Job Controller* do sucesso ou não da execução.

Os tipos de mensagens enviadas pelos demais componentes da arquitetura BioNimbus que devem ser respondidas pelo *Monitoring Service* estão relacionados com o acompanhamento do estado de *jobs*. São definidos três pares de mensagens. O primeiro par trata da submissão de *jobs* para serem executados na federação de nuvens:

- `JobStartReq`: requisição de submissão de *jobs* a serem executados na federação. Deve conter uma coleção de dados da classe `JobInfo` que indica os serviços, com seus parâmetros, que o usuário deseja que sejam executados;
- `JobStartReply`: resposta do *Monitoring Service* indicando o sucesso na submissão dos *jobs*. Possui as informações contidas na requisição com dados atualizados, como identificador único de cada *job* e seu estado atual.

O segundo par de mensagens se refere ao acompanhamento do estado do *job* na federação de nuvens:

- `JobStatusReq`: requisição de informações sobre o estado de um determinado *job* na federação. Deve conter pelo menos seu identificador único;

- `JobStatusReply`: resposta do *Monitoring Service* sobre o estado atual do *job*, com dados da classe `JobInfo`.

Relacionada ao par de mensagens acima descrito está a mensagem `TaskEnd`. Ela é enviada pelo *plug-in* de integração quando uma *task* finalizar sua execução. Sua definição é:

- `TaskEnd`: notificação sobre a finalização de execução de uma *task*. Deve conter dados da classe `TaskInfo`, como identificador e arquivos de saída.

O último par de mensagens sobre o gerenciamento de *jobs* refere-se ao cancelamento da submissão de um *job* na federação de nuvens:

- `JobCancelReq`: requisição de cancelamento de submissão de um *job* na federação. Deve conter pelo menos seu identificador único;

- `JobCancelReply`: resposta do *Monitoring Service* confirmando o cancelamento da submissão. Contém dados da classe `JobInfo` sobre o *job* cancelado.

Para realizar o monitoramento de todos os pedidos, o *Monitoring Service* envia mensagens do tipo `TaskStatusReq` periodicamente para os *plug-ins* de integração dos provedores de serviço para obter informações sobre o estado de execução das tarefas (*tasks*) correspondentes aos pedidos (*jobs*). Desta forma, ele consegue verificar se uma execução é finalizada com sucesso e sem violar os parâmetros de SLA acordados com o usuário correspondente.

Para conseguir realizar as atividades descritas acima, o *Monitoring Service* deve ser capaz de coletar informações sobre alocação de recursos e sobre execução de tarefas, informações estas que dependem do tipo de aplicação sendo executada [28]. Portanto, é necessário que sejam estabelecidos alguns critérios sobre a frequência em que os dados são obtidos e seus correspondentes formatos. Isso para atender os requisitos de confiabilidade dos dados de monitoramento e de flexibilidade com relação aos dados das variadas aplicações possíveis.

Para a arquitetura BioNimbus, é previsto que as implementações enviem não só a intervalos regulares as mensagens de coleta de dados para os membros da federação reconhecidos como provedores, mas também sempre que a necessidade por dados atuais seja crítica. Para que essas coletas de dados tenham o menor consumo possível de banda de rede, é sugerido o uso de *timestamps* lógicos para cada tipo de informação, indicando a "versão"atual daquela informação, de forma que apenas dados modificados trafeguem entre os *plug-ins* e o *Monitoring Service*. Para atender o requisito de flexibilidade, a exemplo do *Discovery Service*, podem ser utilizados formatos como o JSON para troca de informações.

Ademais, para uma federação de nuvens, o *Monitoring Service* deve possuir outras características, como [19]:

- Escalabilidade, para lidar com um grande número de recursos e tarefas a serem monitoradas;

- Elasticidade, para lidar com adições e remoções frequentes de recursos e serviços de maneira transparente; e obviamente
- Federação, para lidar com a entrada e saída de provedores de maneira flexível.

Para atender esses requisitos, é proposto o uso de uma infraestrutura de indexação de recursos descentralizada, que poderia ser a mesma DHT disponível para o *Discovery Service*.

Scheduling Service

O ***Scheduling Service*** recebe os pedidos de execução de tarefas — na arquitetura Bio-Nimbus são chamados de *jobs* — e distribui dinamicamente as instâncias desses pedidos — chamadas de *tasks* — entre os provedores de serviço que fazem parte da federação, mantendo um registro das execuções escalonadas, controlando a carga em cada provedor, e redistribuindo as execuções quando recursos estão sobrecarregados.

Antes do *job* ser distribuído a um provedor por meio da mensagem `TaskStartReq` para se tornar efetivamente uma tarefa em execução (*task*), ele é escalonado de acordo com uma política configurada. Cada política deve implementar a seguinte interface:

```
Map<JobInfo, PluginInfo> schedule(List<JobInfo> jobsList);
```

Dessa maneira, cada política recebe uma lista de *jobs* a serem escalonados e retorna um mapeamento dos *jobs* aos *plug-ins* para onde devem ser enviados para execução. Para realizar esta tarefa, as políticas tem acesso às informações obtidas pelo *Discovery Service*. Para isso, o *Scheduling Service* envia mensagens do tipo `CloudReq`, recebendo a resposta em uma mensagem do tipo `CloudReply`. As políticas também precisam levar em conta os parâmetros de SLA negociados com o usuário que solicitou a execução. Essas informações são enviadas pelo *Monitoring Service* juntamente com o pedido (*job*).

Além de receber *jobs* e escaloná-los, juntamente com sua função de acompanhar seu estado de execução, realizada por meio do envio de mensagens do tipo `TaskStatusReq`, o *Scheduling Service* é responsável por verificar se um determinado *job* está há muito tempo aguardando, o que pode indicar um provedor sobrecarregado ou com poucos recursos. Nesses casos, ele poderá enviar mensagens de cancelamento (`TaskCancelReq`) para o atual provedor do *job* e escaloná-lo novamente.

O *Scheduling Service* aguarda por mensagens de escalonamento originadas no *Monitoring Service* e envia uma mensagem de resposta, com dados sobre o resultado do escalonamento. Essas mensagens são:

- `JobSchedReq`: requisição de escalonamento de *jobs* na federação. Deve conter uma coleção de dados da classe `JobInfo` que indica quais serviços foram requisitados e seus parâmetros, juntamente com os dados de SLA negociados com o usuário;
- `JobSchedReply`: resposta do *Scheduling Service* indicando o resultado do escalonamento. Além do identificador do *job* escalonado, possui informações da classe `PluginInfo` indicando os dados do provedor selecionado.

Storage Service

O ***Storage Service*** é responsável por coordenar a estratégia de armazenamento dos arquivos a serem consumidos ou produzidos pelas tarefas executadas na arquitetura BioNimbus, decidindo sobre a distribuição e replicação dos arquivos entre os diferentes provedores.

Para realizar essa função, primeiramente, o *Storage Service* tem acesso às informações sobre a federação enviando ao *Discovery Service* mensagens do tipo `CloudReq`. Com isso, o serviço saberá as condições atuais de armazenamento de cada um dos provedores que fazem parte da federação.

Quando um pedido de armazenamento de arquivo for feito, a decisão do local para onde efetivamente será feito o *upload* do arquivo é realizada pela estratégia configurada por meio da seguinte interface:

```
PluginInfo chooseStorage(FileInfo file);
```

Uma estratégia de armazenamento deve implementar esta interface de forma que o *Storage Service* repasse as informações sobre o arquivo (dados em *FileInfo*) e ela retorne o provedor de infraestrutura (dados em `PlugInfo`) para onde será feito o *upload* do arquivo baseado em algoritmo próprio e os dados obtidos do *Discovery Service*. Assim, a arquitetura BioNimbus possibilita a utilização de diferentes estratégias de armazenamento.

Em adição, é esse serviço que mantém uma tabela com os arquivos armazenados na federação, com seus respectivos identificadores e localização. Essa tabela pode ser persistida de uma ou mais maneiras, chamadas `FileTables`. A cada armazenamento confirmado por um *plug-in* de integração após um *upload* de sucesso, o *Storage Service* percorre sua coleção de tabelas acionando em cada uma delas a interface:

```
void storeFile(ID id, FileInfo file, List<PluginFileInfo> pluginFiles);
```

Cada tabela, então, é responsável por mapear o identificador de um arquivo às suas informações originais e aos locais onde ele está armazenado.

Para realizar essas suas funções, o *Storage Service* receberá três grupos de mensagens, as quais deverá atender. O primeiro grupo trata do processo de decisão do local de armazenamento de arquivo:

- `StoreReq`: requisição de escolha do local de armazenamento de um arquivo. Deve conter dados da classe `FileInfo` para que sejam usados pela estratégia de armazenamento;
- `StoreReply`: resposta do *Storage Service* indicando o local de armazenamento para o *peer* que solicitou o armazenamento. Deve conter, além dos dados da requisição, dados da classe `PluginInfo`;
- `StoreAck`: mensagem enviada por um *plug-in* de integração após a finalização do *upload* de um arquivo à sua infraestrutura. Ela contém dados da classe `PluginFileInfo` que serão armazenados nas tabelas de arquivo do *Storage Service*.

O segundo grupo de mensagens se refere a consultas sobre os arquivos armazenados na federação de nuvens:

- `ListReq`: requisição de listagem de arquivos armazenados na federação de nuvens. Ela não contém nenhum conteúdo adicional;
- `ListReply`: resposta do *Storage Service* com uma coleção de dados da classe `FileInfo` que representa a consolidação de todos os arquivos mantidos na federação naquele momento.

O último grupo de mensagens enviadas ao *Storage Service* se refere à indicação do local de armazenamento de um arquivo, a partir do qual poderia ser feito um *download*:

- `GetReq`: requisição sobre o local de onde pode ser feito o *download* de um dado arquivo. Deve conter pelo menos o identificador único do arquivo;
- `GetReply`: resposta do *Storage Service* com dados da classe `PluginInfo` sobre o provedor para onde deve ser enviada uma mensagem do tipo `FilePrepReq` para iniciar o processo de *download*.

Para definir estratégias eficientes de armazenamento para uma federação de nuvens para bioinformática, é necessário, primeiro, ter em mente as características de dados biológicos, entre elas:

- Dados biológicos normalmente ocupam enormes volumes de armazenamento;
- Durante execuções de ferramentas de bioinformática, não há atualização simultânea de dados por mais de um usuário;
- Fragmentação e replicação podem ser feitos de diferentes modos, dependendo da aplicação; e
- Proveniência dos dados (*data provenance*) é um conceito essencial.

Considerando essas características e o ambiente de federação de nuvens, uma alternativa é a utilização de bancos de dados NoSQL (*Not only SQL*), tais como o HBase [47], o Cassandra [49] ou o MongoDB [18], pois além de suas características específicas para ambiente em nuvem e de larga escala, possuem características que permitem a conjugação de dados (arquivos biológicos) com conjuntos de informações (proveniência de dados).

Para as funcionalidades de replicação e de fragmentação, podem ser projetados mecanismos que estejam em uma camada acima da infraestrutura de armazenamento. Tais mecanismos devem levar em conta parâmetros como formato dos dados, tamanho de armazenamento disponível, taxa de transferência da rede, localização geográfica, entre outros, com o intuito de diminuir a quantidade de dados transferidos entre membros da federação e aumentar o paralelismo nas execuções das aplicações.

Fault Tolerance Service

Em ambientes de computação em nuvem, falhas em máquinas ocorrem e é bem reconhecido o fato de que essas falhas sejam antes uma regra do que exceção. Logo, uma federação de nuvens deve ser projetada levando em consideração requisitos de tolerância a falhas e alta disponibilidade.

O ***Fault Tolerance Service*** é responsável por garantir que todos os demais serviços controladores do núcleo da arquitetura estejam disponíveis. Para isso, ele mantém um registro dos *peers* da rede P2P nos quais cada um dos serviços está executando, e monitora sua execução por meio de mensagens de *heartbeat*, de intervalo curto.

Ao detectar que algum *peer* não responde após um tempo, o *Fault Tolerance Service* inicia um algoritmo de eleição para o serviço indisponível que define em qual *peer* será iniciada uma nova instância. Ao fim deste processo, todos os *peers* são avisados dessa escolha.

Para o ambiente de federação de nuvens, um requisito importante é a escalabilidade no mecanismo de detecção de falhas. Por isso, na arquitetura BioNimbus é necessária a utilização de um mecanismo que suporte um grande número de membros da federação. Além disso, é preciso que o processo de eleição seja descentralizado, para não criar interdependência entre membros da federação.

Para a detecção de falhas de maneira escalável, pode ser usado um mecanismo baseado em um algoritmo *Gossip* [91], em que os diversos membros da federação enviam listas de *peers* conhecidos com valores de *heartbeat* atualizados para os demais. Aqueles *peers* que não tiverem seus valores acrescidos depois de algum tempo são reconhecidos como falhos por todos os demais.

Para eleição de um novo responsável por algum serviço controlador, deve ser utilizada uma estratégia de coordenação distribuída. Um exemplo é o protocolo de *broadcast* atômico (*atomic broadcast protocol*) [76]. O sistema *open-source* Apache Zookeeper [43] é uma implementação desse protocolo. Ele é capaz de executar uma eleição distribuída atômica após a detecção de uma falha sem haver discordância entre diferentes *peers* da rede.

4.2.3 Serviços de Interação com o Usuário

Os serviços de interação com o usuário são os responsáveis por coletar as ações as quais os usuários desejam fazer na federação de nuvens, e repassá-las em formato de mensagens para o núcleo da arquitetura BioNimbus. Eles podem ser de vários tipos: páginas *web*, linhas de comando, interface gráficas, sistemas gerenciados de *workflows*, etc.

Para isso, os serviços precisam inicialmente fazer parte da rede P2P. Uma vez conectados, eles devem enviar as mensagens previstas pelos serviços controladores do núcleo da arquitetura BioNimbus para realizar suas tarefas, como listar os arquivos armazenados na federação, fazer um *upload* de arquivo, ou submeter um *job* para execução.

A Seção 4.3 apresenta de forma detalhada os principais casos de uso previstos para a arquitetura BioNimbus.

4.3 Casos de Uso e Troca de Mensagens

A arquitetura BioNimbus foi projetada para atender as ações comumente executadas pelos pesquisadores durante a realização de um projeto genoma. Para isso, é importante ressaltar que, tipicamente, as ferramentas de bioinformática utilizadas possuem características semelhantes, as quais são:

- Um conjunto de **arquivos de entrada** que serão analisados pela ferramenta, que podem ser originados a partir da saída da máquina de sequenciamento ou de outra ferramenta de bioinformática;
- Um conjunto de **arquivos de saída** contendo o resultado da análise realizada pela ferramenta;
- Um conjunto de **parâmetros de execução** que podem influenciar, por exemplo, na utilização dos recursos de *hardware* ou no formato dos arquivos de entrada e de saída.

Sendo assim, a arquitetura prevê uma série de casos de uso que tentam se adequar às características listadas acima. O objetivo é prover um conjunto completo de ações que possam ser realizadas sobre a arquitetura BioNimbus, de forma que os serviços de interação com o usuário sejam capazes de atender às necessidades dos pesquisadores.

A seguir, são listados e descritos esses casos de uso e o fluxo de mensagens entre os componentes da arquitetura para que cada um deles seja executado a contento. Em seguida, é apresentado um exemplo completo de execução de uma ferramenta de bioinformática, bastando para isso concatenar alguns dos casos de uso descritos.

4.3.1 *Upload* de Arquivos

Para que seja possível executar uma ferramenta de bioinformática na federação de nuvens computacionais construída pela arquitetura BioNimbus, é preciso que os arquivos de entrada consumidos pela ferramenta estejam armazenados na própria federação. Além disso, um *upload* necessita ser realizado quando um processamento for finalizado e os arquivos de saída precisarem ser armazenados em uma infraestrura que não a do processamento. Portanto, é necessário um caso de uso para o envio de arquivos para a federação. Assim, o *upload* de arquivos é feito nos seguintes passos:

1. O serviço de interação utilizado pelo usuário ou o *plug-in* de integração coleta informações sobre o arquivo, e envia uma mensagem do tipo `StoreReq` para o *Storage Service*;
2. O *Storage Service* decide o local de armazenamento usando a estratégia configurada, e responde para o serviço de interação ou *plug-in* com uma mensagem do tipo `StoreReply`;
3. O serviço de interação ou *plug-in* envia uma mensagem do tipo `FilePutReq` para o *plug-in* de integração do provedor escolhido para o armazenamento;

4. Este *plug-in*, após receber o arquivo completamente, inicia o envio do arquivo para o armazenamento local de sua infraestrutura;

5. Ao fim deste processo, ele envia uma mensagem do tipo `StoreAck` para o *Storage Service* e uma do tipo `FilePutReply` para quem enviou o arquivo, confirmando o armazenamento para ambos.

Quando o último passo for finalizado, será possível consultar o *Storage Service* e obter o identificador único do arquivo para sua utilização em futuras submissões de *jobs* para a federação. A Figura 4.2 mostra a sequência de mensagens no caso de um *upload* de arquivo realizado por um usuário.

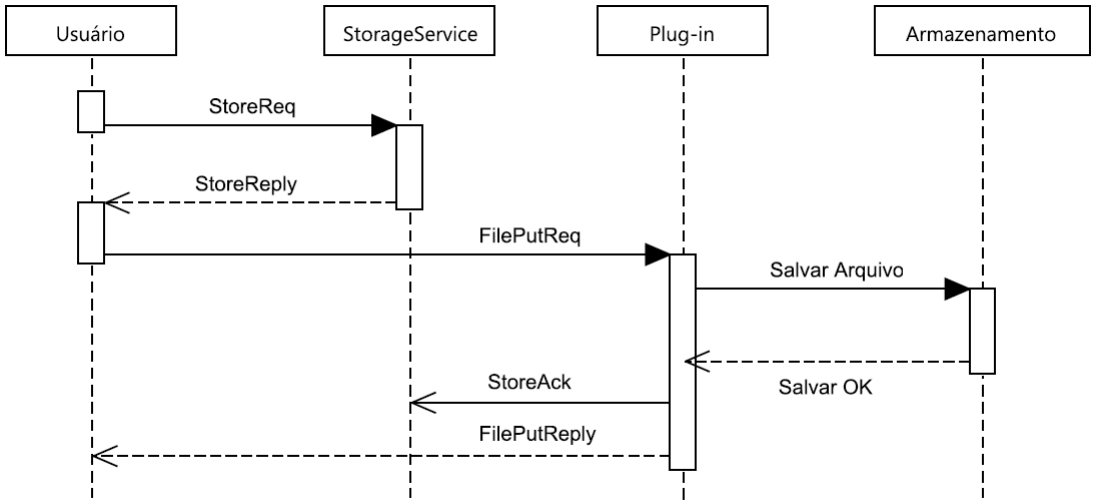

Figura 4.2: Sequência de mensagens para *upload* de arquivos por um usuário.

4.3.2 Listagem de Arquivos

A listagem de arquivos é necessária para se obter, principalmente, os identificadores únicos dos arquivos armazenados na federação de nuvens. Estes identificadores são gerados durante o processo de *upload* de arquivos e são usados para referenciar estes mesmos arquivos durante a submissão de *jobs* para a federação. A listagem de arquivos é feita nos seguintes passos:

1. O serviço de interação com o usuário envia uma mensagem do tipo `ListReq` pela rede P2P para o *StorageService*;

2. O *Storage Service* responde com uma mensagem do tipo `ListReply`;

3. O serviço de interação, com os dados recebidos, é responsável por apresentá-los ao usuário, principalmente os identificadores dos arquivos.

A Figura 4.3 representa graficamente a troca de mensagens para realizar uma listagem de arquivos armazenados na federação de nuvens.

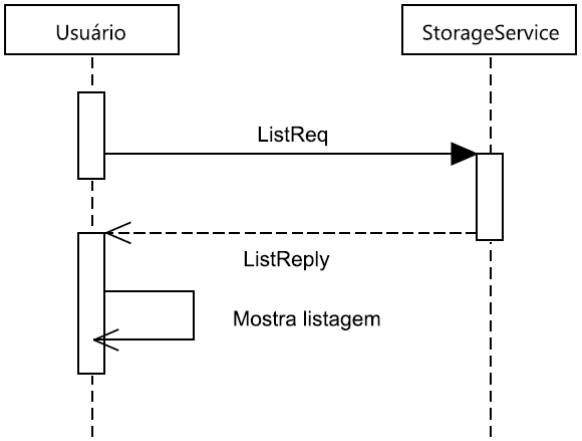

Figura 4.3: Sequência de mensagens para listagem de arquivos por um usuário.

4.3.3 *Download* de Arquivos

O *download* de arquivos, na arquitetura BioNimbus, ocorre em dois momentos. Primeramente, quando o usuário desejar ter acesso ao resultado de um processamento. E em segundo lugar, quando algum *plug-in* de integração necessitar ter acesso a um arquivo armazenado em outra infraestrutura para ser usado como entrada em um processamento realizado na infraestrutura do seu respectivo provedor. O *download* de arquivos pode ser feito seguindo os seguintes passos:

1. O serviço de interação com o usuário ou o *plug-in* de integração de um provedor de infraestrutura envia uma mensagem do tipo `GetReq` para o *StorageService*, contendo o identificador do arquivo desejado;

2. O *Storage Service* consulta suas tabelas e responde com uma mensagem do tipo `GetReply`, indicando com dados da classe `PluginInfo` onde o arquivo pode ser obtido;

3. Com a indicação do local, quem iniciou o processo agora envia uma mensagen do tipo `FilePrepReq` para o *plug-in* indicado;

4. O *plug-in* realiza seus preparativos, que pode ser baixar o arquivo de um armazenamento distribuído para a máquina onde ele mesmo executa, e envia a resposta do tipo `FilePrepReply`, indicando que está pronto para o *download*;

5. Quem iniciou o processo envia agora uma mensagem do tipo `FileGetReq` para o *plug-in*;

6. O qual responde na mensagem `FileGetReply` com o conteúdo do arquivo.

A Figura 4.4 mostra como se dá a troca de mensagens para que o *download* de um arquivo aconteça.

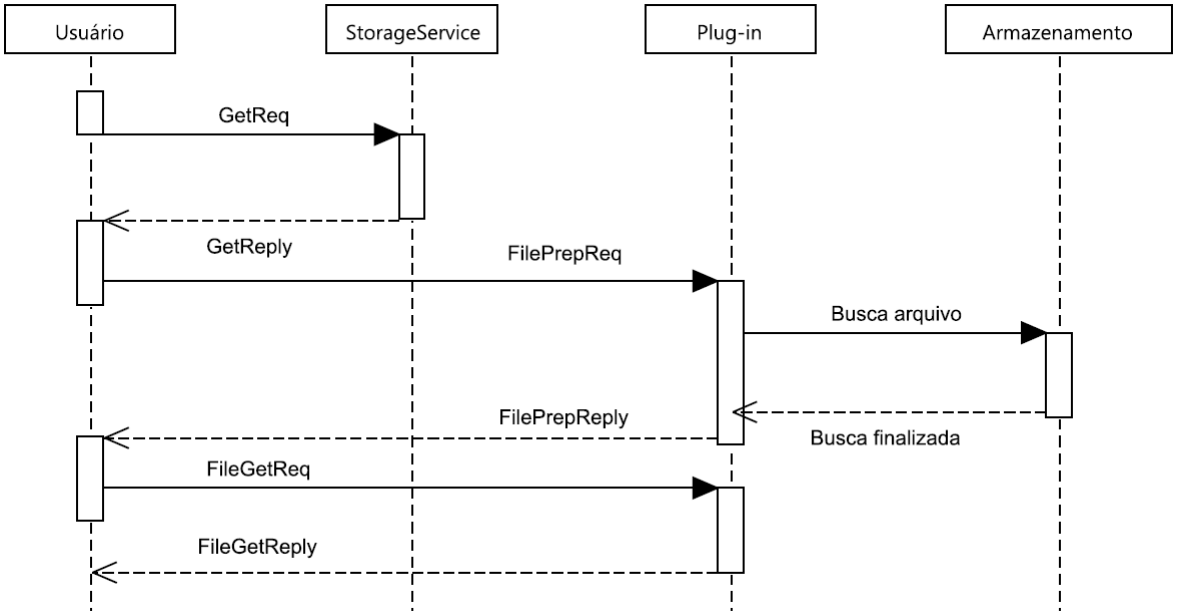

Figura 4.4: Sequência de mensagens para *download* de arquivos por um usuário.

4.3.4 Submissão de *Jobs*

A submissão de *jobs* é a atividade central para a arquitetura BioNimbus. Nela o usuário indica qual aplicação deve ser executada e quais seus parâmetros. Cabe ao serviço de interação com o usuário coletar os dados necessários de forma que seja possível o envio para a federação. O envio pode ser de um ou mais *jobs* em uma única submissão. Vários *jobs* podem ser enviados quando se deseja a execução ao mesmo tempo de diferentes aplicações ou da mesma aplicação com diferentes parâmetros. Isto pode ocorrer quando o usuário estiver utilizando um serviço de gerenciamento de *workflows*, por exemplo. Vale ressaltar que a arquitetura não prevê qualquer relação entre *jobs* que forem submetidos simultaneamente, comportando-se como se cada um deles fosse enviado de forma individual.

A submissão de *jobs* pode ser realizada nos seguintes passos:

1. O serviço de interação com o usuário, após coletar as informações sobre os *jobs*, envia-os à federação por meio do *Job Controller*;

2. Este faz a verificações necessárias junto ao *Security Service* e repassa as informações até agora obtidas para o *SLA Controller*;

3. Após a negociação de SLA, esse controlador, juntamente com os dados do acordo, envia uma mensagem do tipo `JobStartReq`, contendo uma coleção de dados da classe `JobInfo`, para o *Monitoring Service* por meio da rede P2P;

4. O *Monitoring Service* cria o registro do *job* em sua estrutura de dados para monitoramento e requisita o escalonamento ao *Scheduling Service* por meio de uma mensagem do tipo `JobSchedReq`;

5. O *Scheduling Service* aciona a política de escalonamento configurada para cada um dos *jobs* enviados;

6. Com a indicação de onde serão executados, para cada *job* ele envia uma mensagem do tipo `TaskStartReq` com seus dados para o *plug-in* de integração da infraestrutura escolhida para execução;

7. O *plug-in*, por sua vez, cria uma nova *task* vinculada ao *job* enviado, e responde com uma mensagem do tipo `TaskStartReply` para o *Scheduling Service* que usará os dados da classe `TaskInfo` para consultar seu andamento, fazer reescalonamentos, se necessário, e informar o *Monitoring Service* do local de execução;

8. Com os dados sobre as *tasks* recebidos a partir do *plug-in*, o *Monitoring Service* repassa os dados ao *SLA Controller* por meio de uma mensagem do tipo `JobStartReply`, indicando sucesso na submissão, cumprindo o acordo negociado;

9. Enquanto isso, o *plug-in* verifica a existência de arquivos de entrada para a *task* que deve ser iniciada e envia mensagens do tipo `GetReq` para o *Storage Service*, iniciando o processo de *download* dos arquivos, já descrito na Seção 4.3.3;

10. Ao fim do processo de *download* dos arquivos de entrada, o *plug-in* inicia efetivamente a execução da *task* em sua infraestrutura e aguarda sua finalização;

11. Quando a execução acabar, o *plug-in* inicia o processo de *upload* dos arquivos de saída, como descrito na Seção 4.3.1;

12. Depois de armazenar os arquivos de saída, o *plug-in* notifica o *Monitoring Service* com uma mensagem do tipo `TaskEnd`.

A Figura 4.5 mostra graficamente a troca de mensagens realizada durante a submissão, a execução e a finalização de um *job*.

4.3.5 Consulta de *Jobs*

A consulta de *jobs* é feita em dois níveis. No primeiro, o usuário deseja saber, via *Job Controller* qual é o estado de um *job* que submeteu. No outro, o *Monitoring Service* deseja saber o estado de execução de uma *task* que foi escalonada. Para o primeiro caso, os passos são os seguintes:

1. O *Job Controller*, em favor do usuário, envia uma mensagem do tipo `JobStatusReq`, com o identificador do *job*, para o *Monitoring Service*, por meio da rede P2P;

2. O *Monitoring Service* envia ao serviço do usuário a resposta `JobStatusReply` com o estado do *job*.

No segundo caso, os passos são os seguintes:

1. O *Monitoring Service* envia uma mensagem do tipo `TaskStatusReq` para o *plug-in* de integração com o identificador da *task*;

2. Que responde com uma mensagem `TaskStatusReply` com os dados sobre a *task*.

A Figura 4.6 mostra graficamente uma sequência de mensagens com a combinação dos dois casos acima expostos, na qual uma consulta sobre um *job* feita por um usuário inicia um processo de consulta sobre a *task* correspondente no *Monitoring Service*.

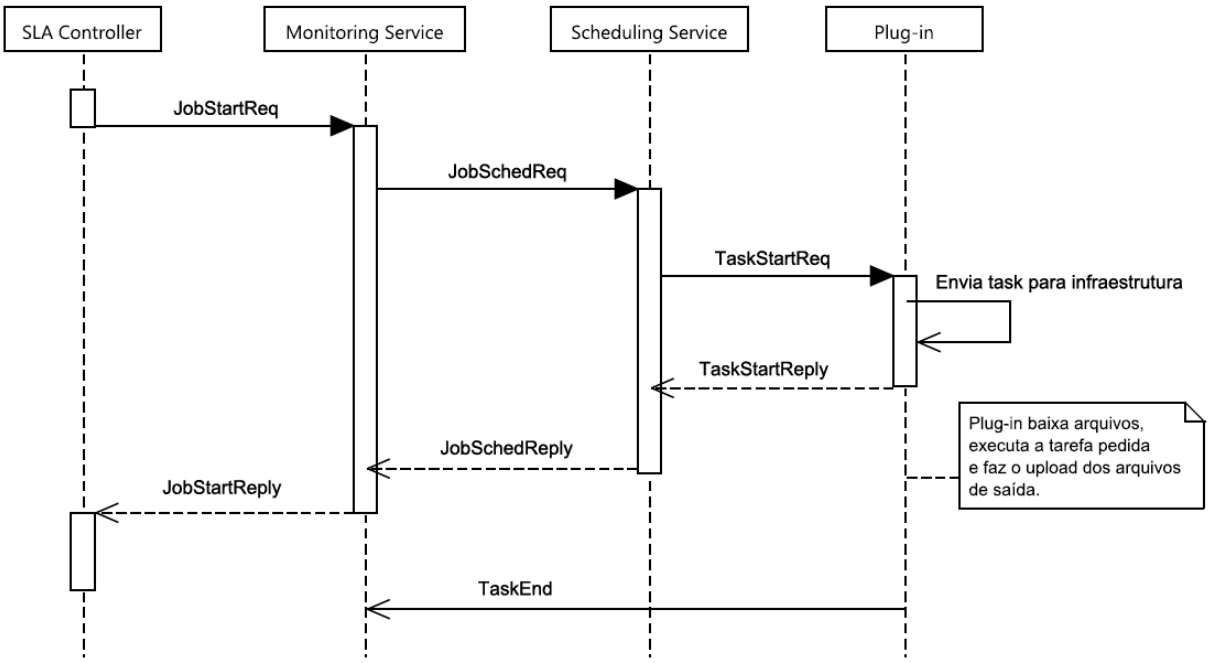

Figura 4.5: Sequência de mensagens para a submissão, a execução e a finalização de um *job* por um usuário.

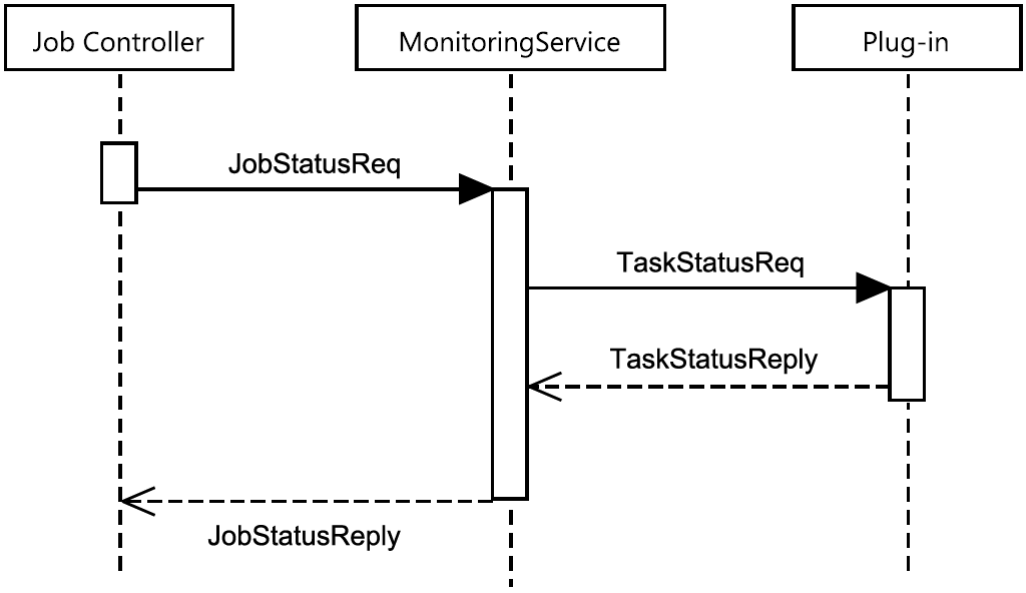

Figura 4.6: Sequência de mensagens para consulta de um *job* por um usuário.

4.3.6 Cancelamento de *Jobs*

O cancelamento de *jobs* pode ser feito quando um usuário quiser retirar um *job* da fila de escalonamento ou interromper sua execução na federação. A arquitetura também prevê o cancelamento de *tasks* por parte do *Scheduling Service* quando este desejar mover a execução de um *job* de uma infraestrutura para outra.

O cancelamento de *jobs* pode ser feito nos seguintes passos:

1. O *Job Controller*, a pedido do usuário, envia uma mensagem do tipo `JobCancelReq` por meio da rede P2P para o *Monitoring Service* contendo o identificador do *job*;
2. O *Monitoring Service* verifica se existem *tasks* referentes ao *job*. Para cada uma delas, o serviço envia uma mensagem do tipo `TaskCancelReq` para o *plug-in* de integração da infraestrutura onde a *task* está executando, contendo seu identificador;
3. Após interromper a execução, o *plug-in* responde ao *Monitoring Service*, confirmando o cancelamento com uma mensagem `TaskCancelReply`;
4. Quando tiver recebido a resposta de todos os *plug-ins*, o *Monitoring Service* confirma o cancelamento do *job* para o usuário com uma mensagem do tipo *JobCancelReply*.

No cancelamento de *tasks* para efeitos de refazer o escalonamento, o *Scheduling Service* realiza os passos 2 e 3 acima. A Figura 4.7 mostra graficamente a troca de mensagens necessária para o cancelamento de um *job*.

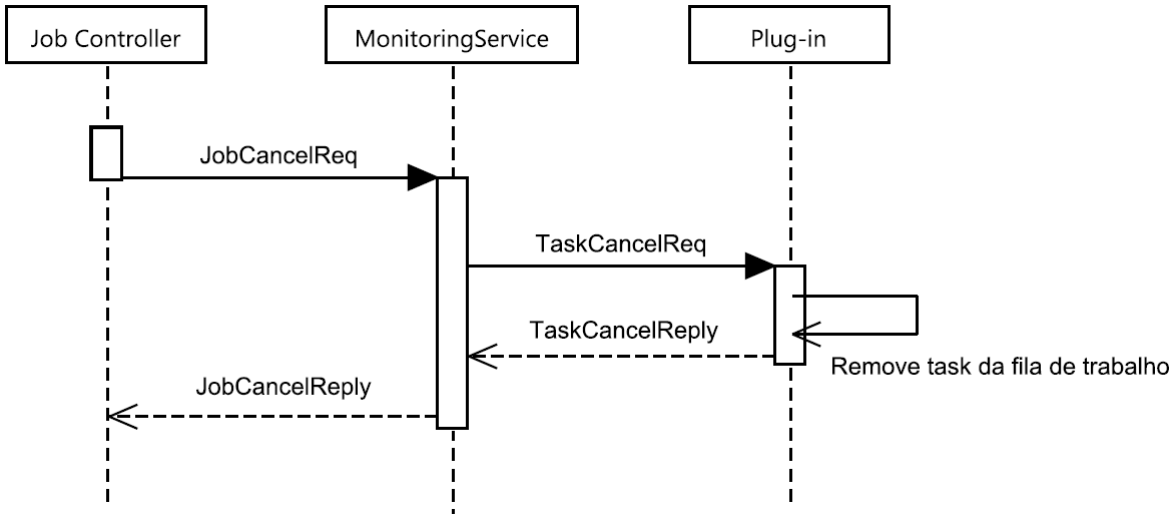

Figura 4.7: Sequência de mensagens para cancelamento de um *job* por um usuário.

4.3.7 Exemplo de Submissão e Execução de um *Job* na Federação

Nesta seção será apresentado um exemplo da aplicação de alguns dos casos de uso descritos anteriormente em uma situação real de utilização da federação de nuvens computacionais para a execução de ferramentas de bioinformática. Os passos descritos abaixo estão representados graficamente na Figura 4.8.

Inicialmente (passo 1), o usuário interage com a arquitetura BioNimbus por meio de uma interface, a qual poderia ser uma linha de comando ou uma interface *web*, por exemplo. Nesse momento, o usuário informa detalhes da aplicação ou do *workflow*, e essas informações são enviadas para o *Job Controller* em forma de *jobs* a serem executados.

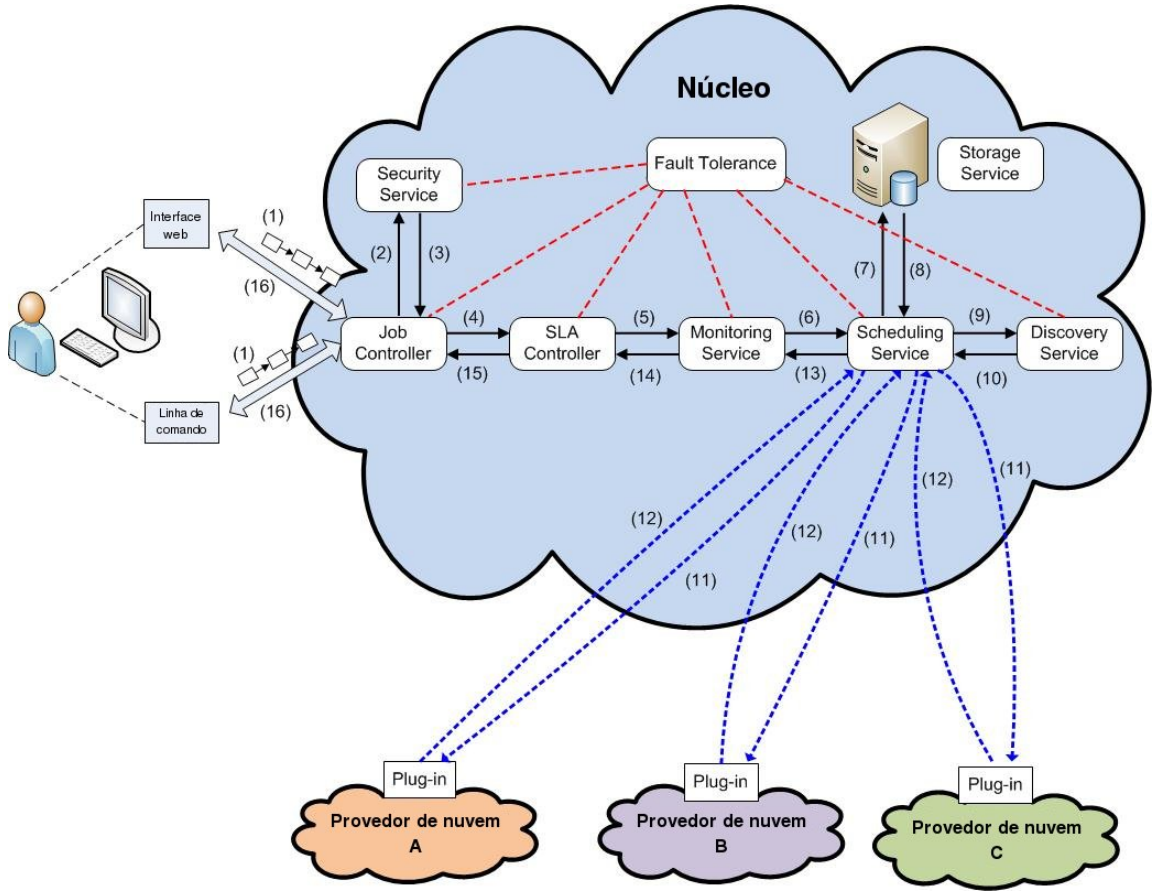

Figura 4.8: Execução de *jobs* na arquitetura BioNimbus.

Depois, o *Job Controller* repassa as informações para que o *Security Service* (passo 2) faça a verificação das permissões do usuário para acessar os recursos da federação, o qual responde (passo 3) de acordo.

Se o usuário possuir permissão para utilizar recursos da federação, então uma mensagem é enviada para o *SLA Controller* (passo 4), o qual registra o *template* SLA fornecido inicialmente pelo usuário, dando início ao ciclo de vida do acordo. Após isso, o controlador submete os *jobs* para o *Monitoring Service* (passo 5), juntamente com o SLA, a fim de que eles sejam executados nos termos especificados. O *Monitoring Service* guarda essas informações em uma lista de *jobs* pendentes e solicita ao *Scheduling Service* (passo 6) um escalonamento de acordo com os parâmetros de SLA.

Para fazer sua escolha de acordo com a política de escalonamento configurada, o *Scheduling Service* verifica a existência e a localização dos arquivos de entrada com ajuda do *Storage Service* (passos 7 e 8) e requisita dados sobre os provedores para o *Discovery Service* (passos 9 e 10). Com esses dados, o *Scheduling Service* executa o algoritmo de escalonamento da política configurada e encaminha cada *job* para o provedor escolhido (passos 11 e 12).

O *Scheduling Service* então informa ao *Monitoring Service* (passo 13) o sucesso no escalonamento nos termos do SLA, para que ele possa monitorar o estado do *job* até sua finalização, verificando se há algum descumprimento no acordo. Ao fim da execução, o *Monitoring Service* envia uma mensagem para o *SLA Controller* (passo 14) para que este

finalize o ciclo de vida do SLA. Depois disso, o *SLA Controller* repassa a mensagem para o *Job Controller* (passo 15), terminando todo o processo, que registra os dados finais sobre a execução. Eles estarão disponíveis para consulta pelo usuário (passo 16).

4.4 Discussão

A arquitetura BioNimbus é projetada de forma que atenda os objetivos, e cumpra os requisitos de uma federação de nuvens computacionais, listados na Seção 2.2. Além disso, a proposta é diferente das arquiteturas apresentadas na mesma Seção.

4.4.1 Requisitos Atendidos

Na arquitetura BioNimbus, os objetivos de alcançar a impressão de recursos ilimitados, de permitir a eliminação da dependência de único provedor, e de otimizar o uso de recursos dos provedores federados, são atingidos por meio da criação de uma rede P2P de diversos provedores, com o gerenciamento destes por parte dos serviços controladores da federação. Com isso, tem-se uma união de recursos computacionais, oferecidos por diversos provedores, que permitem sua utilização por parte de todos os provedores e por parte de usuários externos que tenham acesso à federação, sendo possível a distribuição desse uso entre seus membros.

Com o *Discovery Service*, é cumprido o requisito de automatismo na identificação das nuvens que compõem a federação. Isso porque ele monitora os membros da rede P2P, sua entrada e saída, e também faz requisições para identificar os recursos e aplicações disponíveis neles. Assim, de maneira transparente e automática, o serviço constrói uma visão geral da federação, que pode ser consultada por todos seus membros.

Por meio do uso combinado do *Scheduling Service*, do *Storage Service* e do *Monitoring Service*, a arquitetura é capaz de cumprir os requisitos de previsão de carga de aplicações e mapeamento de serviços a recursos. As escolhas de distribuição de processamento e de armazenamento realizadas pelos dois primeiros serviços podem ser baseadas nas informações sobre tempo de execução, localização de dados e carga de utilização de recursos, entre outras, recolhidas durante o tempo pelo *Monitoring Service*. Essas informações podem ser organizadas de tal forma que seja possível criar uma combinação aplicação-localização de dados-infraestrutura possivelmente adequada para cada escolha.

Finalmente, a utilização do *Security Service* e dos *plug-ins* de integração permitem à arquitetura construir um modelo de segurança interoperável, já que o serviço cria uma política geral para a federação, mas que respeita as diversas políticas particulares de cada uma das nuvens.

4.4.2 Comparação com Outras Propostas

À semelhança das demais propostas da Seção 2.2, a arquitetura aqui apresentada prevê um componente que esteja localizado dentro da infraestrutura das nuvens que são membros da federação, que neste caso é o *plug-in* de integração. Contudo, sua função é coletar dados sobre a nuvem, como recursos, aplicações e políticas próprias, e apresentá-las à federação em formato padrão, e também oferecer uma interface para a utilização desses recursos e aplicações, baseadas em suas políticas. Não é sua função executar serviços de

escalonamento e descobrimento, ao contrário das outras propostas. Essa escolha foi feita pois procurou-se dar uma visão única a todos os membros da federação com o objetivo de evitar conflitos ou escolhas baseadas em dados diferentes, que poderiam gerar, por exemplo, um balanceamento de carga ineficiente.

Além disso, a escolha acima foi tomada pois na arquitetura BioNimbus o usuário não interage diretamente com uma das nuvens, mas utiliza um serviço de interação previsto pela arquitetura, como também acontece com a proposta de Buyya *et al.* [15]. A intenção é uniformizar o acesso à federação, facilitando sua utilização por parte dos usuários.

Capítulo 5

Estudo de Caso

Neste capítulo, é descrito o estudo de caso realizado como prova de conceito para a arquitetura BioNimbus. Nele foram federados diferentes provedores de serviço, privados e públicos, para realizar uma federação de provedores de aplicações de bioinformática. Os provedores são detalhados na Seção 5.1. Com esta federação, foi executado um *workflow* de bioinformática com ferramentas e dados reais, descrito na Seção 5.2. A federação foi construída por meio de um protótipo da arquitetura BioNimbus, detalhado na Seção 5.3. O protótipo contém os serviços controladores principais para execução de *workflows*: *Discovery*, *Storage*, *Monitoring* e *Scheduling*. Ademais, foram utilizados *plug-ins* de integração implementados também durante este trabalho. A interação, no estudo de caso, é feita por meio de uma linha de comando. Finalmente, a comunicação foi feita por meio da implementação de um módulo de mensageria com uma camada de roteamento acima para a construção de redes P2P, detalhado na Seção 5.4. A discussão sobre os resultados do estudo de caso é feita na Seção 5.5.

5.1 Ambiente de Execução

Para o estudo de caso, foram utilizados dois *clusters* distintos:

- Na Universidade de Brasília, um *cluster* Hadoop [36] foi implementado com três máquinas, cada uma possuindo um processador Intel Core 2 Duo com 2.66GHz de frequência, 4 gigabytes de memória RAM e sistema operacional Linux, distribuição Ubuntu 11.10. No total, o *cluster* possui seis núcleos (*cores*) de processamento e 565 gigabytes de armazenamento. O armazenamento foi implementado por meio do Hadoop Distributed Filesystem (HDFS) [13]. As aplicações de bioinformática executam no *cluster* utilizando o modo *streaming* do Hadoop MapReduce [23].

- Na infraestrutura da *Amazon Elastic Compute Cloud* (EC2) [57] também foi implementado um *cluster* Hadoop. Foram utilizadas cinco máquinas virtualizadas, cada uma com processador Intel Xeon com 2.27GHz de frequência, 8 gigabytes de memória RAM e sistema operacional Linux, distribuição Fedora 8. Uma das máquinas servia como controladora do *cluster*, enquanto as demais eram nós trabalhadores. O *cluster* totalizava então oito núcleos de processamento, com 1,6 terabytes de armazenamento. O armazenamento e a execução das aplicações se deram nos mesmos moldes do *cluster* da UnB.

5.2 *Workflow*, Ferramentas e Dados

O *workflow* de bioinformática escolhido para o estudo de caso tem como objetivo identificar genes diferencialmente expressos em células humanas cancerosas do rim e do fígado [62, 77], com fragmentos sequenciados por sequenciadores Illumina [44]. O *workflow* consiste em quatro fases, como mostra a Figura 5.1.

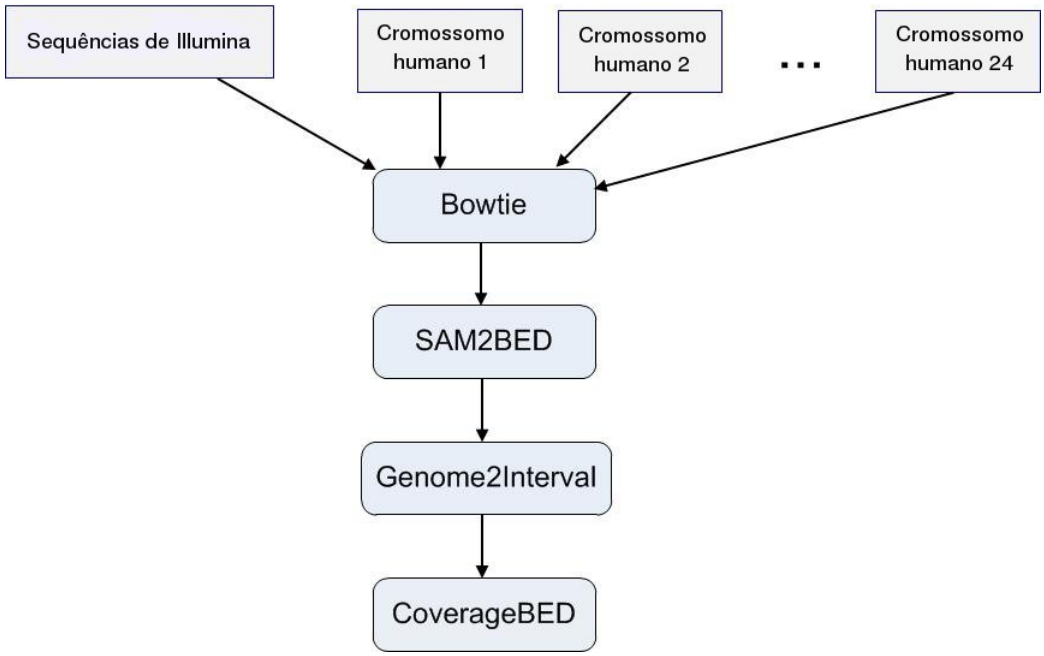

Figura 5.1: *Workflow* utilizado para identificar o nível de expressão de genes em células cancerosas do rim e do fígado.

Na primeira fase, de mapeamento, os fragmentos são mapeados aos 24 cromossomos humanos de referência. O objetivo é identificar a região do genoma de referência onde cada fragmento está localizado. Um conjunto de fragmentos mapeados na mesma região permitem inferir que elas possuem a mesma organização estrutural do genoma de referência. A ferramenta utilizada para fazer esse mapeamento é a ferramenta Bowtie [52]. Na segunda fase, o formato SAM de saída do mapeamento é convertido para o formato BED com um *script* de conversão chamado *sam2bed*, implementado exclusivamente para esse *workflow*. Na terceira fase, com um *script* chamado *genome2interval*, são gerados intervalos de tamanho fixo baseados no tamanho de cada cromossomo que serão utilizados na fase seguinte. Nesta fase, a partir das saídas da segunda e da terceira fase, são gerados histogramas que indicam o número de fragmentos mapeados por intervalo dos cromossomos com a ferramenta coverageBED da *suite* BEDtools [71].

Vale lembrar que as quatro aplicações mencionadas acima eram oferecidas como serviço pelos dois provedores de serviço utilizados no estudo de caso. Além disso, os 24 cromossomos do genoma de referência humano (HG19) foram obtidos no banco de dados do NCBI [66] no endereço `ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/Assembled_chromosomes/seq/`. A descrição do genoma pode ser obtida em `http://www.ncbi.nlm.nih.gov/genome/assembly/293148/`.

5.3 Protótipo da Arquitetura

Para o estudo de caso, foi utilizada uma implementação prototipal da arquitetura, desenvolvida durante a realização deste trabalho, utilizando a linguagem Java, compatível com o JDK 1.6.

A seguir, é descrita mais detalhadamente a implementação de cada item da arquitetura realizada para o estudo de caso. Uma representação gráfica do protótipo pode ser vista na Figura 5.2.

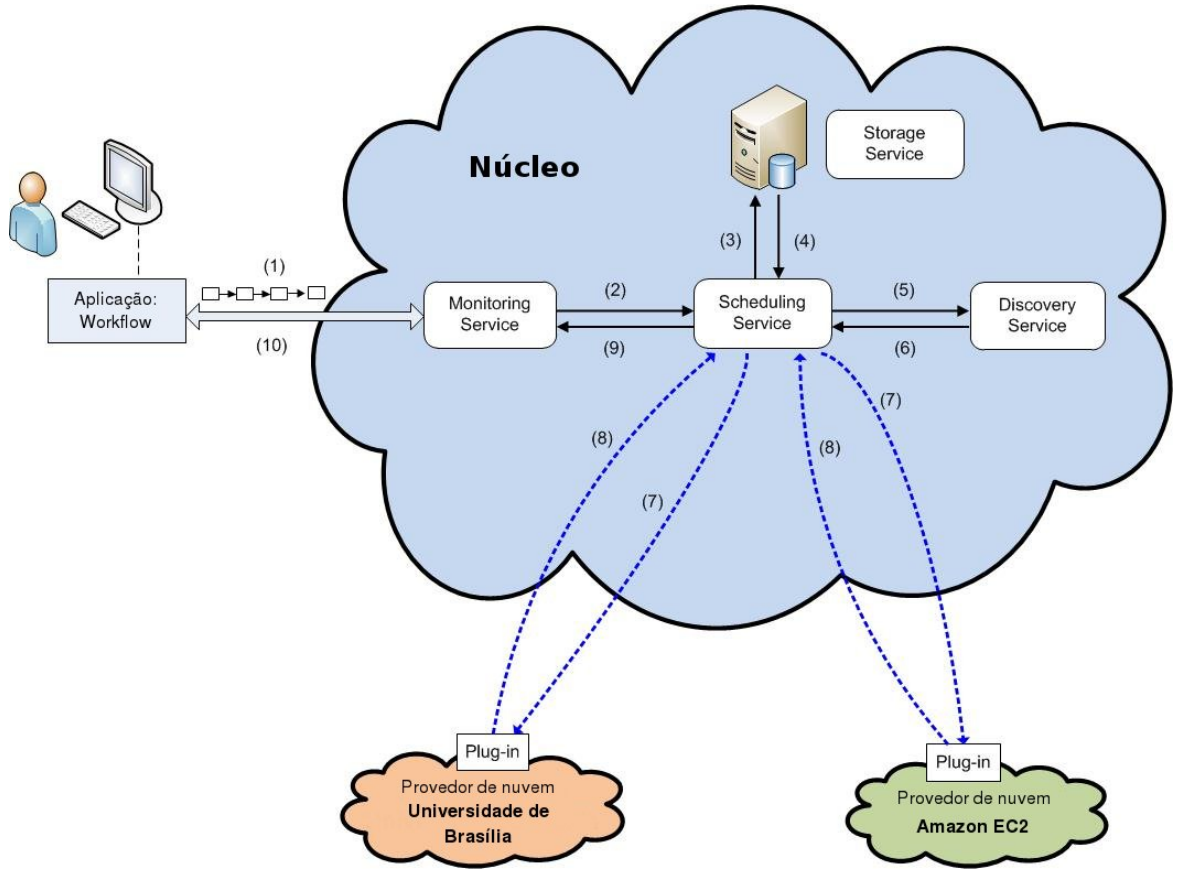

Figura 5.2: Protótipo implementado para estudo de caso mostrando serviços controladores e provedores utilizados.

5.3.1 *Discovery Service*

A implementação do *Discovery Service* possui duas *threads* de execução. A primeira é responsável por fazer a atualização e a limpeza da estrutura de dados na qual são armazenadas as informações sobre as infraestruturas federadas. A segunda *thread* aguarda por mensagens vindas pela rede P2P que devem ser tratadas pelo serviço.

Para armazenar as informações das infraestruturas foi utilizada uma estrutura de dados do tipo `map`, na qual a informação da infraestrutura está mapeada com seu identificador único. Além disso, cada dado desses possui um *timestamp* para indicar o momento do último mapeamento. Para atualizar a infraestrutura, a primeira *thread* de execução é

acionada a intervalos de 30 segundos para enviar as mensagens do tipo `InfoReq` para todos os membros da rede P2P. Este intervalo foi escolhido pois em experimentos com valores menores a rede P2P e os *peers* — serviços controladores e *plug-ins* — ficavam sobrecarregados. A resposta vinda daqueles que são *plug-ins* de integração, as mensagens do tipo `InfoReply`, são tratadas pela segunda *thread*, que atualiza o mapeamento com as novas informações e seu respectivo *timestamp*. Por fim, a primeira *thread*, a cada acionamento, também verifica aquelas informações que não tiveram sua última data de modificação alterada nos últimos 90 segundos e remove-as do mapeamento, considerando assim que esses *plug-ins* saíram da rede P2P.

Ademais, o tratamento das requisições sobre o estado da federação (`CloudReq`) também é feito pela segunda *thread* de execução, que monta e envia a resposta (`CloudReply`) com o mapeamento mantido pelo serviço.

5.3.2 *Monitoring* e *Scheduling Service*

No protótipo para prova de conceito da arquitetura, o *Monitoring Service* e o *Scheduling Service* foram implementados de maneira unificada, como apenas um processo em execução. Para realizar o seu trabalho de registro, escalonamento e acompanhamento de *jobs* enviados pelos usuários da federação, a implementação utiliza três estruturas de dados principais, do tipo `map`. São elas:

- `PendingJobs`: mapeamento do identificador do *job* requisitado para suas respectivas informações, representando aqueles *jobs* aguardando serem (re)escalonados;
- `RunningJobs`: mapeamento do identificador da *task* em execução para suas informações e para as informações do *job* ao qual ela faz parte;
- `CancelingJobs`: mapeamento do identificador da *task* em execução para o identificador do *job* ao qual ela faz parte e para o cliente que solicitou o cancelamento.

No serviço existe uma *thread* de execução responsável por aguardar as requisições e as respostas vindas dos demais componentes da arquitetura. Ao receber uma requisição para início de *job(s)* (`JobStartReq`), esta *thread* gera um identificador único para cada *job* e salva as informações no mapa `PendingJobs`. Após isso, ela aciona a política de escalonamento configurada por meio da interface estabelecida pela arquitetura BioNimbus, a qual retorna o mapeamento dos *jobs* com o *plug-in* de integração para onde cada um deles deve ser enviado para execução.

Em seguida, uma vez os *jobs* devidamente escalonados, a mesma *thread* envia as requisições para criação de *tasks* (`TaskStartReq`) nas infraestruturas e aguarda as respostas correspondentes (`TaskStartReply`). A cada chegada de resposta, o serviço então remove o *job* do mapa `PendingJobs` e cria uma entrada no mapa `RunningJobs`, com as informações do *job* e as informações da *task* em execução na infraestrutura, somente sendo removida ao fim de sua execução.

A segunda *thread* de execução existente na implementação, acionada em intervalos de 15 segundos, é responsável por realizar o acompanhento dos *jobs* recebidos pelo serviço. Primeiramente, ela envia requisições de estado (`TaskStatusReq`) para cada um dos *jobs* cadastrados em `RunningJobs`. A resposta (`TaskStatusReply`), tratada pela *thread* descrita anteriormente, pode iniciar um processo de reescalonamento caso o serviço assim o

decida. Mensagens de cancelamento (`TaskCancelReq` e `TaskCancelReply`) serão trocadas e o *job* será reinserido no mapa `PendingJobs` e removido de `RunningJobs`.

A segunda *thread* também tem a tarefa de verificar se existem *jobs* pendentes em `PendingJobs` e iniciar um novo processo de escalonamento, já descrito, e além disso enviar mensagens de consultas para o *Discovery* (`CloudReq`) e o *Storage Services* (`ListReq`), cujas respostas (`CloudReply` e `ListReply`) serão recebidas pela primeira *thread* e utilizadas pela política de escalonamento sempre que necessário.

Nos experimentos com o protótipo, foi utilizado o algoritmo DynamicAHP [12]. A idéia principal do algoritmo é o mapeamento dos recursos disponíveis dos provedores de serviço aos pedidos de execução baseado no processo de tomada de decisão proposta por Saaty [78]. As variáveis utilizadas para o escalonamento estão relacionadas ao tamanho do arquivo de entrada para a execução e ao poder computacional de cada provedor de serviço.

5.3.3 *Storage Service*

A implementação do *Storage Service* também utiliza duas *threads* de execução para realizar seu trabalho. A principal aguarda as requisições enviadas pelos demais componentes da arquitetura. Para tratar a requisição de salvamento de arquivos (`StoreReq`), o serviço aciona a estratégia de armazenamento configurada por meio da interface estabelecida pela arquitetura. A única estratégia implementada para o presente estudo de caso executa um algoritmo *round-robin* entre os *plug-ins* de integração que indicam possuir armazenamento suficiente para o arquivo que pretende-se salvar na federação.

Após o local de armazenamento ter sido escolhido, a resposta correspondente (`StoreReply`) é enviada para quem iniciou a requisição, que procederá o envio do arquivo para o destino indicado pelo *Storage Service*. Ao fim dessa operação, o *plug-in* de integração no qual foi feito o armazenamento do arquivo envia a mensagem de confirmação prevista pela arquitetura (`StoreAck`). Ela contém os dados necessários para que o serviço seja capaz de manter a tabela de arquivos armazenados na federação de nuvens.

Para o estudo de caso, foi implementado um *backend* simples para a manutenção da tabela de arquivos. Sempre que uma nova confirmação de armazenamento chega ao *Storage Service*, ele adiciona uma entrada em um mapa do identificador do arquivo para suas informações (nome, tamanho, local de armazenamento, etc.). Ao mesmo tempo, esse mapeamento é gravado em um arquivo na máquina onde o serviço executa, no formato JSON [22]. Sempre que o serviço inicia sua execução, ele verifica a existência desse arquivo e carrega na memória o último estado da tabela de arquivos contida nele.

Os outros dois tipos de requisição tratados pela *thread* principal são de listagem (`ListReq`) e localização (`GetReq`) de arquivos. No primeiro caso, a *thread* constrói a resposta (`ListReply`) com o mapeamento que está atualmente na memória. No segundo, ela constrói a resposta (`GetReply`) buscando a informação de localização no mapeamento por meio do identificador fornecido na requisição.

Finalmente, a *thread* de execução secundária é acionada em intervalos de 30 segundos para requisitar (com a mensagem `CloudReq`) a situação atual da federação ao *Discovery Service*. A informação obtida é utilizada pelo algoritmo da estratégia de armazenamento configurada.

5.3.4 *Plug-ins* de Integração

Por fim, foi desenvolvido um protótipo de *plug-in* de integração para o Apache Hadoop, igualmente em Java. Ele foi utilizado para incluir os provedores na federação e para receber requisições de transferência de arquivos e de execução de aplicações.

O processo de execução do *plug-in* está dividido da seguinte maneira:

- Uma *thread* principal, responsável por aguardar por eventos da rede P2P. Os eventos tratados são aqueles relacionados ao trabalho do *plug-in*: pedidos de informação sobre o provedor (`InfoReq`), pedidos de execução, de cancelamento e de informação de execuções (`TaskStartReq`, `TaskCancelReq` e `TaskStatusReq`), e pedidos de *upload* e de *download* de arquivos (`FilePreqReq`, `FileGetReq` e `FilePutReq`);

- Um *pool* de *threads* trabalhadoras, responsáveis por tratar os eventos recebidos pela *thread* principal. Por ele são realizados os acessos ao *cluster* Hadoop. Esses acessos podem ser de acesso ao sistema de arquivos distribuído (HDFS), de execução de tarefas (*tasks*) no modelo MapReduce e de busca de informações sobre o estado do *cluster*;

- Um *thread* auxiliar para coleta do resultado das atividades do item anterior, e para construção das respostas ao requisitante dos dados obtidos, executada a intervalos regulares.

Para a troca de informações entre as *threads* descritas acima, são usadas algumas estruturas de dados. As principais são:

- Fila com requisições pendentes recebidas pela *thread* principal. Ela é consumida pelas *thread* trabalhadoras;

- Lista de tarefas (*tasks*) em execução no *cluster* Hadoop, com informações sobre o estado de cada tarefas;

- Lista de tarefas em fase de finalização, ou seja, aguardando o término da transferência dos arquivos de saída da execução para o armazenamento da federação de nuvens.

5.4 Comunicação na Federação

Como mencionado, a comunicação entre os componentes da federação de nuvens se dá por meio de uma rede P2P. No protótipo, foram implementados dois módulos: um de mensageria e um de P2P. O primeiro é a camada mais baixo nível, responsável pelo tratamento de conexões TCP e pelo envio e recebimento de mensagens e de arquivos. Esse módulo trata as mensagens de maneira genérica, sem conhecimento de seus detalhes. O segundo módulo, de P2P, é implementado no topo do módulo de mensageria, e especifica as mensagens do protocolo P2P implementado, e contém as estruturas de dados para a construção da rede P2P e para o roteamento das mensagens entre os *peers*.

5.4.1 Módulo de Mensageria

Para a comunicação entre os componentes da arquitetura, foi desenvolvido um serviço de mensageria utilizando a biblioteca de comunicação Netty [45], responsável pela gerência de eventos das conexões TCP. Os envios ocorrem assincronamente, isto é, os usuários do módulo encaminham mensagens ao módulo, sem necessitar aguardar pelo resultado. Somente se houver um erro, há uma notificação sinalizada a quem originou o envio. Da mesma forma, o recebimento de mensagens também é assíncrono, ou seja, o serviço usuário do módulo se registra, indicando quais tipos de mensagem aguarda, e é notificado posteriormente em caso de recebimento, sem a necessidade de bloquear sua execução para esperar por novas mensagens.

Além disso, o módulo trata de maneira transparente a diferenciação entre envios de mensagens e de arquivos, usando o mesmo canal de comunicação (mesma porta TCP). O objetivo é eliminar a necessidade de regras complexas de *firewall*. Essa diferenciação é feita internamente ao módulo, verificando os primeiros *bytes* da transferência. Se eles forem os primeiros *bytes* de um requisição (`GET` ou `PUT`) ou de um resposta do protocolo HTTP, então a transferência passa a ser tratada como uma transferência de arquivos. Caso contrário, é um envio de uma das mensagens estabelecidas pelo protocolo P2P e pela arquitetura BioNimbus.

As mensagens enviadas por meio do módulo de mensageria são serializadas e desserializadas por interfaces genéricas (`encode()` e `decode()`, respectivamente), eliminando a necessidade do módulo conhecer detalhes sobre a mensagem. A serialização é feita em formato JSON [22], com a ajuda da biblioteca Jackson [20]. A biblioteca consegue, a partir de um objeto Java, gerar o texto em formato JSON com os dados correspondentes, e também consegue, a partir de dados JSON, construir o objeto Java correspondente.

5.4.2 Módulo para Protocolo P2P

Para o estudo de caso foi implementada uma versão simplificada do protocolo *Chord* [85]. O módulo é responsável por identificar novos *peers* na rede, incluindo-os no anel Chord, o qual é a estrutura de dados para guardar informações sobre os *peers*, como a latência de rede e o tempo pertença (*uptime*). Uma vez que um *peer* é identificado, o módulo envia mensagens de *ping* constantemente, para se certificar de sua presença na rede. Caso não haja uma resposta após certo tempo, o módulo considera que o *peer* abandonou a rede, retirando-o do anel. Finalmente, o módulo é responsável também por rotear as mensagens entre os serviços da arquitetura, isto é, ao receber mensagens ou arquivos, e ao perceber remoção de *peers* da rede, o módulo gera eventos para os demais componentes da arquitetura que estão em uma camada acima.

5.5 Resultados e Discussão

O *workflow* apresentado na Seção 5.2 foi executado em cada uma das nuvens apresentadas (UnB e Amazon EC2) individualmente e, em seguida, na federação montada com as duas nuvens. No total, para cada um dos casos, foram executados quatros *jobs* (quatro fases do *workflow*) para cada um dos 24 cromossomos humanos, totalizando então 96 *jobs*. Os tempos de execução para as três execuções realizadas estão indicados na Tabela 5.1.

Tabela 5.1: Tempo de execução do *workflow* em cada nuvem e na federação.

Provedores	hora:minuto:segundo
Universidade de Brasília (UnB)	1:11:47
Amazon EC2	1:18:44
Ambos (UnB and EC2)	1:09:07

Os tempos apresentados não consideram o tempo de transferência inicial dos arquivos de entrada (os 24 cromossomos e as SRS) para o armazenamento da federação. Isto significa que as execuções foram realizadas partindo-se do pressuposto que os arquivos de entrada já estavam disponíveis na federação. Para o caso de execução com os dois provedores, os arquivos de entrada estavam localizados no armazenamento da Amazon EC2.

O primeiro ponto a se notar dos valores apresentados na Tabela 5.1 é que não houve piora no tempo de execução, o que poderia ocorrer se houvesse impacto no uso da arquitetura. Este impacto poderia ser do próprio controle realizado pela arquitetura ou do tempo em que arquivos de entrada e de saída estiveram em trânsito entre os provedores. Pelo resultado obtido, mesmo que haja algum impacto, o ganho no aumento do paralelismo se destaca.

Entretanto, a melhora apresentada no tempo de execução ao ser utilizada a federação de dois provedores não foi significativa. Isso contraria resultados anteriores obtidos com o uso da arquitetura BioNimbus na avaliação do algoritmo DynamicAHP [12], quando houve ganho considerável ao ser utilizada uma federação. A principal diferença entre os dois experimentos é que no experimento anterior as nuvens usadas na federação estavam geograficamente próximas umas das outras e a taxa de transferência dos arquivos de entrada era muito alta, tornando o tempo de transferência praticamente irrelevante se comparado ao tempo total de execução.

Por essa razão, é preciso analisar como o tempo de transferência dos arquivos de entrada afetou o tempo total de execução. Este tempo se refere ao tempo de transferência dos arquivos de entrada de um determinado *job* entre o provedor onde os arquivos estão armazenados e o provedor onde o *job* será executado. Na Tabela 5.2 e na Figura 5.3 são apresentados os tempos de execução totais e o tempo de transferência dos arquivos de entrada dos 18 *jobs* do *workflow* com maior tempo de execução individual, juntamente com a porcentagem da transferência na relação com o total. Pode-se notar que a transferência dos arquivos teve grande impacto no tempo total de execução, com um mínimo de 50% de relevância. Coincidentemente, os 18 *jobs* da Tabela 5.2 são aqueles com os maiores arquivos de entrada. Esta análise indica que, no caso de federações de nuvens para execução de *workflows* de bioinformática, os serviços de armazenamento e de escalonamento (na arquitetura BioNimbus, o *Storage Service* e o *Scheduling Service*) devem ser especialmente projetados de tal maneira que minimizem ao máximo a transferência de arquivos de entrada com grandes tamanhos.

Para confirmar essa análise, é importante um estudo de como o uso de uma federação afeta o tempo de execução de cada *job* em comparação com o tempo em execuções com apenas um provedor. A Tabela 5.3 mostra a distribuição de *jobs* por intervalo de tempo de execução em cada um dos três casos executados durante o estudo de caso. O dado mais importante a ser verificado é a diminuição do tempo de execução da maioria dos

Tabela 5.2: Tempo total de execução e tempo de transferência dos arquivos de entrada, com a relação entre ambos.

Tempo total (segundos)	Tempo de transferência (segundos)	Porcentagem do tempo de transferência em relação ao tempo total
4230.297	2114.996	50.0%
4123.492	2264.552	54.9%
4098.571	2337.454	57.0%
4030.492	2297.580	57.0%
3807.501	2229.992	58.6%
3145.645	2168.201	68.9%
3113.729	2116.199	68.0%
3066.488	2058.771	67.1%
3032.701	2018.942	66.6%
3001.165	2137.157	71.2%
2952.875	2087.761	70.7%
2849.506	2074.117	72.8%
2801.489	2023.309	72.2%
2680.382	1892.002	70.6%
2587.076	2006.842	77.6%
2579.184	1959.727	76.0%
2533.254	1928.888	76.1%
2405.470	1899.626	79.0%

jobs. Esse fato pode ser percebido com mais facilidade com a ajuda do histograma da Figura 5.4.

Essa redução do tempo de execução dos *jobs* individualmente se dá principalmente por dois fatores. Primeiramente, quando se executa um *workflow* como o do estudo de caso somente com uma infraestrutura, cria-se uma fila de espera muito extensa por conta do grande número de *jobs* enviados simultaneamente. Grande parte dos *jobs* precisam aguardar durante bom tempo até serem finalmente executados. Ao contrário, com uma federação, essa carga é dividida entre dois ou mais provedores, reduzindo os tempos de espera.

Em segundo lugar, graças à capacidade da arquitetura BioNimbus de monitorar o estado de carga dos provedores e o tempo de espera para executar cada uma das tarefas (*tasks*) enviadas aos provedores, o *Scheduling Service* realiza movimentos de *tasks* de provedores sobrecarregados para provedores mais livres, evitando também, neste caso, longos tempos de espera.

Tabela 5.3: Número de *jobs* executados agrupados por tempo de execução, incluindo tempo de transferência.

Provedores	$t \leq 200$s	$200 < t \leq 1000$s	$t > 1000$s
Universidade de Brasília	34	30	32
Amazon EC2	37	27	32
UnB e Amazon EC2	64	8	24

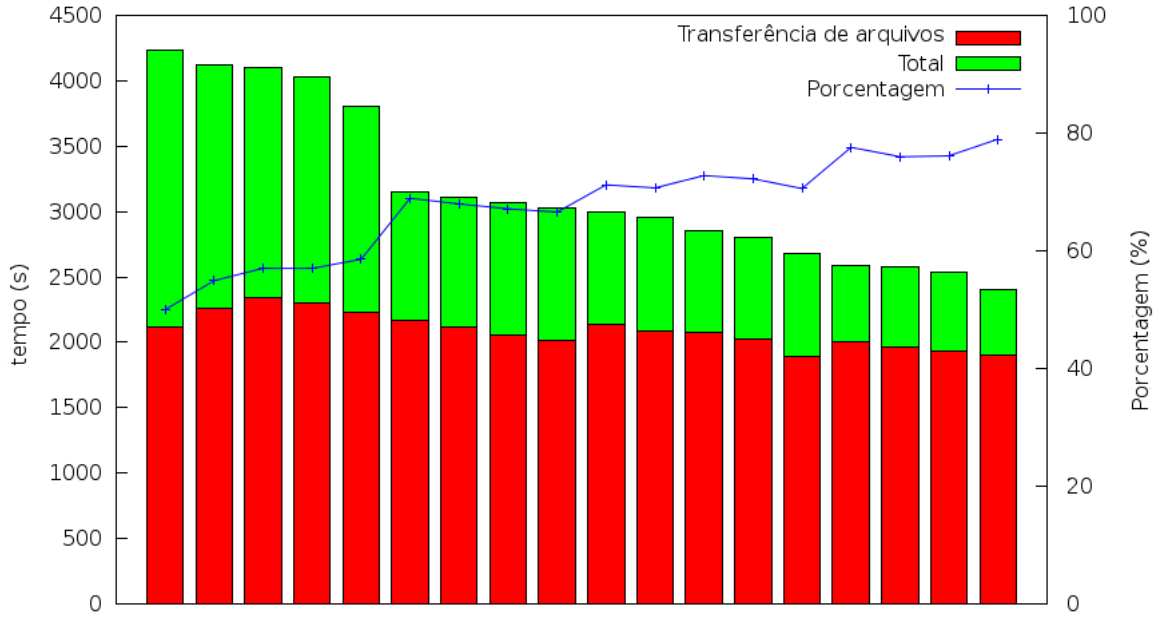

Figura 5.3: Comparação entre tempo total de execução e tempo de transferência de arquivos de entrada dos maiores *jobs* executados na federação. Tempo de transferência está em vermelho e total em azul. A linha azul representa a porcentagem do tempo de transferência em relação ao tempo total.

Ao mesmo tempo, confirmando então a análise anteriormente exposta, é possível perceber por meio da Figura 5.4 e dos valores da Tabela 5.3 que mesmo nas condições descritas acima, certos *jobs* permaneceram com um tempo longo de execução se comparados com os demais. Obviamente, uma causa é que alguns deles atingiram seu tempo ótimo, ou seja, o melhor tempo de acordo com a capacidade de processamento das máquinas utilizadas. Isso ocorre principalmente com aqueles *jobs* que não precisam aguardar transferências de arquivos e foram enviados para provedores livres. Nesse caso, o tempo de execução se explica pelo tamanho do arquivo de entrada. Contudo, outros possuem longo tempo de execução pois existe um longo tempo de transferência de arquivos de entrada. Esta conclusão fica mais clara se for levado em conta o fato de que a arquitetura não considera esse tempo como tempo de espera passível de movimentação de *tasks* entre provedores.

Por fim, existem outros fatores que, após análise do estudo de caso, podem afetar o desempenho da arquitetura BioNimbus na distribuição e execução de *workflows* de bioinformática:

- O escalonador (DynamicAHP) utilizado não considera que *tasks* em fase de preparação, isto é, aguardando a transferência dos arquivos de entrada, ocupem recursos do provedor. Esse fato pode ocasionar sobrecarga em provedores caso muitas *tasks* sejam iniciadas com longo tempo de espera por transferência, mantendo a falsa idéia de que o provedor está *idle*. Sendo assim, escalonadores implementados para o *Scheduling Service* da arquitetura BioNimbus devem ter em mente essa situação.

- Os arquivos de entrada foram baixados todos ao mesmo tempo nos provedores, sem prioridade. Isso faz com que todos as *tasks* dividam a largura de banda disponível

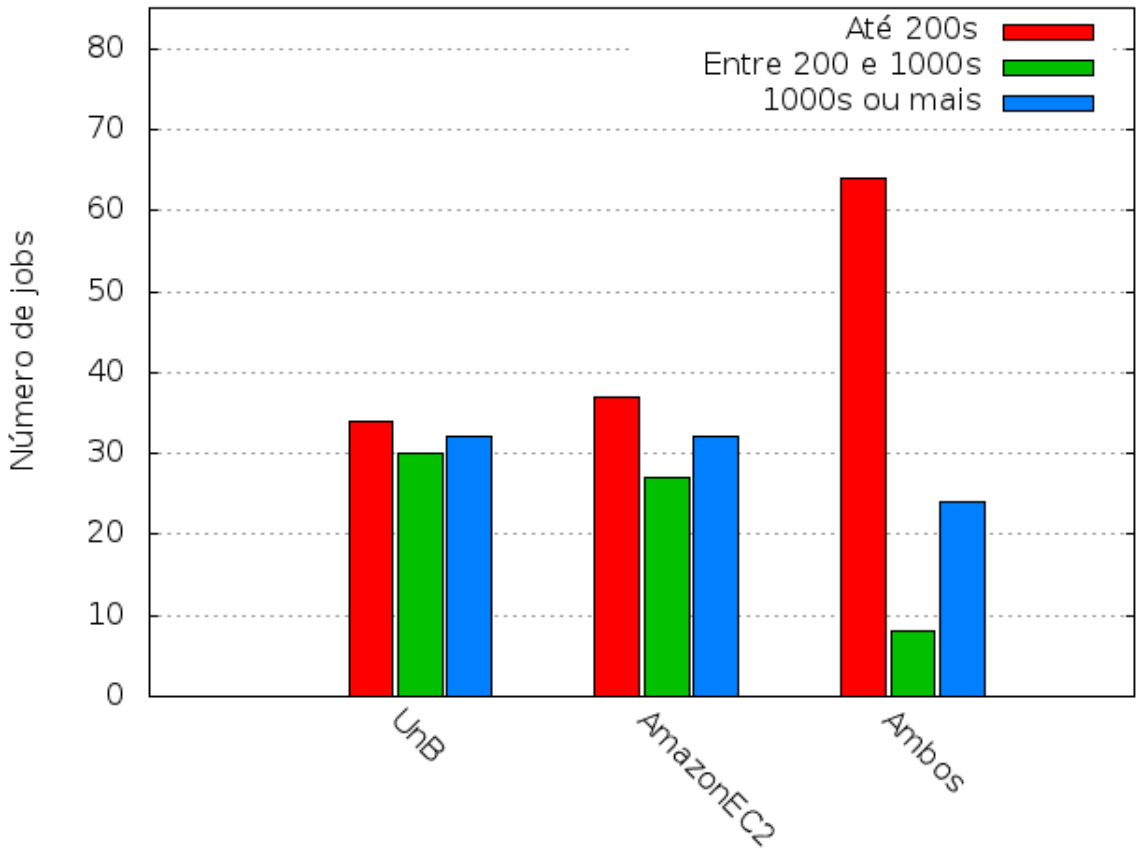

Figura 5.4: Comparação do número de *jobs* agrupados por tempo de execução, incluindo tempo de transferência.

no provedor, podendo prejudicar *downloads* que estavam próximos de seu término caso novos sejam iniciados nesse instante. Por isso, no projeto e na implementação de *plug-ins* de integração para a arquitetura BioNimbus, devem ser estudados os reais efeitos desse problema, e se uma reserva de banda (QoS) deve ser determinada para cada *task* e em quais condições.

Para finalizar a discussão sobre a arquitetura e o estudo de caso, foi visto que a arquitetura BioNimbus realmente é capaz de formar uma federação de nuvens computacionais de maneira transparente ao usuário, pois a execução do *workflow* especificado no estudo de caso foi executado, na visão do usuário, de maneira idêntica nos três casos aplicados. Além disso, foram federadas diferentes nuvens computacionais (pública e privada) e integradas à federação vários tipos de aplicações de bioinformática de maneira flexível.

Capítulo 6

Conclusão e Trabalhos Futuros

No presente trabalho, foi proposta uma arquitetura para federações de nuvens computacionais para execução de *workflows* de bioinformática chamada BioNimbus. Seu objetivo é integrar diferentes infraestruturas computacionais as quais oferecem aplicações de bioinformática de maneira transparente, flexível e tolerante a falhas, e também oferecendo grande capacidade de processamento e armazenamento.

Na especificação da arquitetura, foram identificados os serviços necessários para realizar uma federação de nuvens para bioinformática, com os requisitos necessários a serem cumpridos por implementações. Também foram descritos os casos de uso principais, com a interação entre os serviços e a especificação das mensagens trocadas durante sua execução. Por fim, foi realizada uma comparação entre outras propostas existentes para federação de nuvens com a apresentada por este trabalho.

Além disso, foi realizado um estudo de caso com um protótipo como implementação de referência da arquitetura BioNimbus o qual integrou dois provedores diferentes, público e privado, com o intuito de verificar sua funcionalidade e seu desempenho na prática. Nesse estudo de caso, foi usado um *workflow* para identificação de diferenças de expressão em genes de células cancerosas do rim e do fígado de humanos. Os dados utilizados foram fragmentos produzidos por um sequenciador automática de alto desempenho Illumina e os 24 cromossomos humanos. Para efetivar a análise, o *workflow* foi executado em cada provedor individualmente e depois com os provedores federados por meio da arquitetura BioNimbus.

Com os resultados obtidos, foi possível observar a real aplicabilidade da arquitetura BioNimbus para experimentos com *workflows* de bioinformática. Entre suas características, foram verificadas a sua capacidade real de integrar diferentes infraestruturas e aplicações de bioinformática de maneira transparente, e sua eficiência na distribuição de tarefas entre os provedores federados.

Ao mesmo tempo, também foram apontadas questões que devem ser melhor exploradas em busca de uma melhor aplicação da arquitetura BioNimbus. Entre elas, estão a especificação de melhores estratégias de escalonamento de tarefas e de armazenamento de dados de entrada, de forma a reduzir transferências de arquivos entre provedores de serviços, e o incremento dos *plug-ins* de integração, com o intuito de controlar de maneira mais eficiente essas transferências.

Como trabalhos futuros, primeiramente, é preciso realizar um estudo mais detalhado com diferentes estratégias de armazenamento para redução de transferências de arquivos.

Entre as possibilidades estão o uso de fragmentação e replicação de dados, bem como o mapeamento de tipos de dados com provedores que oferecem aplicações usuárias destes dados. Outro estudo necessário é a identificação das melhores variáveis a serem utilizadas pelo escalonamento, não só considerando tarefas em execução e recursos computacionais disponíveis, mas também tarefas sendo preparadas nos provedores antes mesmo de executarem, bem como variáveis econômicas. Ademais, deve-se procurar formas mais eficientes para controlar a prioridade de transferências de arquivos nos provedores.

Também estão previstas as implementações restantes a fim de completar o protótipo apresentado neste trabalho. Entre elas, estão as implementações do *Fault Tolerance Service*, para realizar tolerância a falhas dos serviços controladores e das execuções iniciadas em provedores faltosos; do *Security Service*, para controlar o acesso de usuários aos provedores e para autenticar membros da federação como provedores válidos; do *Job Controller*, para realizar o controle das execuções de diferentes usuários, cada um com suas variáveis e credenciais, avaliando principalmente sua escalabilidade; e do *SLA Controller*, para se aplicar e monitorar acordos de nível de serviço (SLA), verificando possíveis violações.

Finalmente, vale ressaltar que o presente trabalho resultou em um artigo publicado [79] e um capítulo de livro (*Bioinformatics* — ISBN 980-953-307-202-4) em processo de revisão, com previsão de publicação em agosto de 2012. A arquitetura proposta no trabalho e o protótipo do estudo de caso também fizeram parte de outro artigo publicado [12], no qual serviram como base para um estudo sobre escalonamento de tarefas em ambiente de federação de nuvens.

Referências

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Biologia Molecular da Célula*. Artmed, Porto Alegre, 4ª edition, 2004. viii, 16, 17, 18

[2] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock. Kepler: an extensible system for design and execution of scientific workflows. In *Proceedings of 16th International Conference on Scientific and Statistical Database Management*, pages 423–424, Junho 2004. 20, 26

[3] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, October 1990. 2, 22

[4] S. V. Angiuoli, M. Matalka, A. Gussman, K. Galens, M. Vangala, D. R. Riley, C. Arze, J. R. White, O. White, and W. F. Fricke. CloVR: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing. *BMC Bioinformatics*, 12(356):1–15, 2011. viii, 2, 26, 27

[5] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Fevereiro 2009. 5

[6] R. Arrial, R. Togawa, and M. Brigido. Screening non-coding RNAs in transcriptomes from neglected species using PORTRAIT: case study of the pathogenic fungus Paracoccidioides brasiliensis. *BMC Bioinformatics*, 10(1):239, 2009. 22

[7] H. Balakrishnan, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Looking up data in P2P systems. *Commun. ACM*, 46(2):43–48, 2003. 34

[8] BGI. Bio-Cloud Computing. `http:www.genomics.cn/en/navigation/show_navigation?nid=4143`, 2012. 27

[9] A. Bhandari and M. Singh. 2010 The OAuth 1.0 Protocol. RFC 5849, 2010. 35

[10] I Birol, S. D. Jackman, C. B. Nielsen, J. Q. Qian, R. Varhol, G. Stazyk, R. D. Morin, Y. Zhao, M. Hirst, J. E. Schein, D. E. Horsman, J. M. Connors, R. D. Gascoyne, M. A. Marra, and S. J. M. Jones. De novo transcriptome assembly with ABySS. *Bioinformatics*, 25(21):2872–2877, 2009. 22

[11] T. J. Bittman. The evolution of the cloud computing market. http://blogs.gartner.com/thomas_bittman/2008/11/03/the-evolution-of-the-cloud-computing-market, November 2008. 12

[12] C. A. L. Borges, H. V. Saldanha, E. Ribeiro, M. T. Holanda, A. P. F. Araujo, and M. E. M. T. Walter. Task scheduling in a federated cloud infrastructure for bioinformatics applications. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, CLOSER 2012, pages 114–120, Porto, Portugal, 2012. SciTePress. 56, 59, 64

[13] D. Borthakur. The Apache Software Foundation: HDFS Architecture. http://hadoop.apache.org/common/docs/r0.20.2/hdfs_design.pdf, 2008. 52

[14] K. Breitman. Computação na nuvem. Capítulo apresentado no JAI-2010 do XXX Congresso da SBC, 2010. 7

[15] R. Buyya, R. Ranjan, and R. N. Calheiros. InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services. In *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing*, ICA3PP 2010, pages 13–31. Springer, 2010. viii, 2, 12, 13, 14, 15, 51

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

[17] A. Celesti, F. Tusa, M. Villari, and A. Puliafito. How to Enhance Cloud Architectures to Enable Cross-Federation. In *Proceedings of the 3rd International Conference on Cloud Computing*, IEEE CLOUD 2010, pages 337–345, Miami, Florida, 2010. IEEE Computer Society. viii, 2, 12, 13, 14

[18] K. Chodorow and M. Dirolf. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA, USA, Setembro 2010. 40

[19] S. Clayman, A. Galis, C. Chapman, G. Toffetti, L. Rodero-Merino, L. M. Vaquero, K. Nagin, and B. Rochwerger. Monitoring Service Clouds in the Future Internet. In *Towards the Future Internet - Emerging Trends from European Research*. IOS Press, 2010. 37

[20] Codehaus. Jackson Java JSON-Processor. http://jackson.codehaus.org, Janeiro 2012. 58

[21] Roche Diagnostics Corporation. 454 Life Sciences. http://www.454.com/, Janeiro 2012. 19

[22] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), July 2006. 34, 56, 58

[23] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *6th Conference on Symposium on Operating Systems Design & Implementation*, pages 10–10, Berkeley, CA, EUA, 2004. USENIX Association. 1, 9, 24, 52

[24] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, Nova York, NY, EUA, 2007. ACM. 9

[25] J. C. Dohm, C. Lottaz, T. Borodina, and H. Himmelbauer. Substantial biases in ultra-short read data sets from high-throughput DNA sequencing. *Nucleic Acids Research*, 36(16):e105, 2008. 19

[26] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, July 1998. 22

[27] J. Ekanayake, T. Gunarathene, and J. Qiu. Cloud technologies for bioinformatics applications. *IEEE Transactions on Parallel and Distributed Systems*, 22(6):998–1011, 2011. 26

[28] E. Elmroth and L. Larsson. Interfaces for Placement, Migration, and Monitoring of Virtual Machines in Federated Clouds. In *Proceedings of the 2009 Eighth International Conference on Grid and Cooperative Computing*, GCC '09, pages 253–260, Washington, DC, USA, 2009. IEEE Computer Society. 37

[29] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. 6

[30] M. W. Fiers, A. van der Burgt, E. Datema, J. C. de Groot, and R. C. van Ham. High-throughput bioinformatics with the Cyrille2 pipeline system. *BMC Bioinformatics*, 9(1):96+, Fevereiro 2008. 20

[31] S. A. Filichkin, H. D. Priest, S. A. Givan, R. Shen, D. W. Bryant, S. E. Fox, W. Wong, and T. C. Mockler. Genome-wide mapping of alternative splicing in *Arabidopsis thaliana*. *Genome Research*, 20(1):45–58, 2010. 2, 19

[32] National Center for Biotechnology Information. File format guide. `http://www.ncbi.nlm.nih.gov/books/NBK47537/`, Janeiro 2012. 21

[33] M.P.I. Forum. MPI: A Message-Passing Interface Standard. Technical report, University of Tennessee, Knoxville, Tennessee, Setembro 2009. 11

[34] I Foster. What is the Grid? A Three Point Checklist. *GRIDtoday*, 1(6), Julho 2002. 6, 11

[35] I. Foster, Y. Zhao, I. Raicu, and S. Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop GCE '08*, pages 1–10, 2008. 6, 7, 8, 11

[36] Apache Software Foundation. Apache Hadoop. `http://hadoop.apache.org/`, Janeiro 2012. 1, 9, 24, 52

[37] S. Ghemawat, H. Gobioff, and S. Leung. The Google file system. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 29–43, Nova York, NY, EUA, 2003. ACM. 9

[38] J. Goecks, A. Nekrutenko, J. Taylor, and The Galaxy Team. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biology*, 11(8):R86, 2010. 20, 30

[39] Google. Google App Engine. http://www.google.com/enterprise/cloud/appengine/, Janeiro 2012. 1, 10, 12

[40] Google. Google Docs. `http://docs.google.com/`, Janeiro 2012. 10

[41] D. Hong, A. Rhie, S. Park, J. Lee, Y. S. Ju, S. Kim, S. Yu, T. Bleazard, H. Park, H. Rhee, H. Chong, K. Yang, Y. Lee, I. Kim, J. S. Lee, J. Kim, and J. Seo. FX: an RNA-Seq analysis tool on the cloud. *Bioinformatics*, 28(5):721–723, 2012. 26

[42] D. Hull, K. Wolstencroft, R. Stevens, C. Goble, M. Pocock, P. Li, and T. Oinn. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34(Web Server issue):729–732, Julho 2006. 20, 30

[43] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. Zookeeper: wait-free coordination for internet-scale systems. In *USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 11–11, Berkeley, CA, USA, 2010. USENIX Association. 41

[44] Illumina Inc. Illumina Life Sciences. `http://www.illumina.com/`, Janeiro 2012. 19, 53

[45] Red Hat Inc. Netty - the Java NIO Client Server Socket Framework. `http://www.jboss.org/netty`, Janeiro 2012. 58

[46] L. Jourdren, M. Bernard, M. A. Dillies, and St. Le Crom. Eoulsan: A Cloud Computing-Based Framework Facilitating High Throughput Sequencing Analyses. *Bioinformatics*, 28(11):1542–3, Abril 2012. 26

[47] A. Khetrapal and V. Ganesh. HBase and Hypertable for large scale distributed storage systems. Dept. of Computer Science, Purdue University, http://www.uavindia.com/ankur/downloads/HypertableHBaseEval2.pdf, 2006. 40

[48] K. Krampis, T. Booth, B. Chapman, B. Tiwari, M. Bicak, D. Field, and K. Nelson. Cloud BioLinux: pre-configured and on-demand bioinformatics computing for the genomics community. *BMC Bioinformatics*, 13(1):42+, 2012. 26

[49] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, Abril 2010. 40

[50] B. Langmead, K. Hansen, and J. Leek. Cloud-scale RNA-sequencing differential expression analysis with Myrna. *Genome Biology*, 11(8):R83, 2010. 2, 26

[51] B. Langmead, M. C. Schatz, J. Lin, M. Pop, and S. Salzberg. Searching for SNPs with cloud computing. *Genome Biology*, 10(11):R134, 2009. viii, 2, 24, 25

[52] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25, 2009. 2, 21, 24, 53

[53] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm. What's inside the Cloud? An architectural map of the Cloud landscape. In *Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, CLOUD '09, pages 23–31, Washington, DC, EUA, 2009. IEEE Computer Society. 7

[54] H. Li and R. Durbin. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009. 21

[55] R. Li, Y. Li, X. Fang, H. Yang, J. Wang, K. Kristiansen, and J. Wang. SNP detection for massively parallel whole-genome resequencing. *Genome Research*, 19(6):1124–1132, June 2009. 2, 22, 25

[56] R. Li, H. Zhu, J. Ruan, W. Qian, X. Fang, Z. Shi, Y. Li, S. Li, G. Shan, K. Kristiansen, S. Li, H. Yang, J. Wang, and J. Wang. De novo assembly of human genomes with massively parallel short read sequencing. *Genome Research*, 20(2):265–72, Fevereiro 2010. 22

[57] Amazon Web Services LLC. Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2/`, Janeiro 2012. 1, 10, 25, 52

[58] Amazon Web Services LLC. Amazon Simple Storage Service (Amazon S3). `http://aws.amazon.com/s3/`, Janeiro 2012. 1, 10

[59] B. Ludäscher, I. Altintas, S. Bowers, J. Cummings, T. Critchlow, E. Deelman, D. D. Roure, J. Freire, C. Goble, M. Jones, S. Klasky, T. McPhillips, N. Podhorszki, C. Silva, I. Taylor, and M. Vouk. Scientific Process Automation and Workflow Management. In Arie Shoshani and Doron Rotem, editors, *Scientific Data Management*, Computational Science Series, chapter 13. Chapman & Hall, 2009. 20

[60] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice & Experience - Workflow in Grid Systems*, 18(10):1039–1065, 2006. 20

[61] K. Malde. Flower: extracting information from pyrosequencing data. *Bioinformatics*, 27(7):1041–2, Abril 2011. 21

[62] J. C. Marioni, C. E. Mason, S. M. Mane, M. Stephens, and Y. Gilad. RNA-seq: An assessment of technical reproducibility and comparison with gene expression arrays. *Genome Research*, 18(9):1509–1517, Setembro 2008. 53

[63] P. Mell and T. Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 53(6):50, 2009. 7, 8

[64] Microsoft. Windows Azure. http://www.microsoft.com/windowsazure/, Janeiro 2012. 1, 10, 12

[65] D. L. Nelson and M. M. Cox. *Lehninger princípios de bioquímica*. Sarvier, São Paulo, 3ª edition, 2002. 16, 17

[66] U.S. National Library of Medicine. National center for biotechnology information. `http://www.ncbi.nlm.nih.gov/`, Janeiro 2012. 21, 53

[67] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Scalable Wide-Area Resource Discovery. Technical Report UCB/CSD-04-1334, EECS Department, University of California, Berkeley, 2004. 34

[68] A. Pashalidis and C. J. Mitchell. A taxonomy of single sign-on systems. In *Information Security and Privacy, 8th Australasian Conference, ACISP 2003*, pages 249–264. Springer-Verlag, 2003. 35

[69] B. Pratt, J. J. Howbert, N.I. Tasman, and E. J. Nilsson. MR-Tandem: Parallel X!Tandem using Hadoop MapReduce on Amazon Web Services. *Bioinformatics*, 28(1):136–7, Janeiro 2012. 26

[70] D. Pushkarev, N. F. Neff, and S. R. Quake. Single-molecule sequencing of an individual human genome. *Nature Biotechnology*, 27(9):847–850, August 2009. 2

[71] A. R. Quinlan and I. M. Hall. BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, 26(6):841–842, 2010. 53

[72] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2011. 26

[73] Rackspace. The Rackspace Cloud. http://www.rackspace.com/cloud/, Janeiro 2012. 1

[74] R. Ranjan, L. Chan, A. Harwood, S. Karunasekera, and R. Buyya. Decentralised Resource Discovery Service for Large Scale Federated Grids. In *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, E-SCIENCE '07, pages 379–387, Washington, DC, USA, 2007. IEEE Computer Society. 34

[75] D. Recordon and D. Reed. OpenID 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, DIM '06, pages 11–16, New York, NY, USA, 2006. ACM. 35

[76] B. Reed and F. P. Junqueira. A simple totally ordered broadcast protocol. In *Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, LADIS '08, pages 2:1–2:6, New York, NY, USA, 2008. ACM. 41

[77] M. Robinson and A. Oshlack. A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biology*, 11(3):R25+, 2010. 53

[78] T. L. Saaty. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, 48(1):9 – 26, 1990. 56

[79] H. V. Saldanha, E. Ribeiro, M. Holanda, A. Araujo, G. Rodrigues, M. E. M. T. Walter, J. C. Setubal, and A. Davila. A cloud architecture for bioinformatics workflows. In *Proceedings of 1st International Conference on Cloud Computing and Services Science*, CLOSER 2011, pages 477–483, Noordwijkerhout, Netherlands, 2011. SciTePress. 28, 64

[80] salesforce.com inc. Sales Cloud. `http://www.salesforce.com/crm/sales-force-automation/`, Janeiro 2012. 10

[81] F. Sanger, S. Nicklen, and A. R. Coulson. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences*, 74(12):5463–5467, December 1977. 19

[82] M. C. Schatz. CloudBurst: Highly Sensitive Read Mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, Junho 2009. viii, 2, 24

[83] A. Smith, Z. Xuan, and M. Zhang. Using quality scores and longer reads improves accuracy of Solexa read mapping. *BMC Bioinformatics*, 9(1):128, 2008. 2, 21, 24

[84] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster. Virtual Infrastructure Management in Private and Hybrid Clouds. *IEEE Internet Computing*, 13:14–22, September 2009. http://opennebula.org/. 9

[85] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. Frans Kaashoek, F. Dabek, and H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Transactions on Networking (TON)*, 11(1):17–32, Fevereiro 2003. 28, 58

[86] M. Sultan, M. H. Schulz, H. Richard, A. Magen, A. Klingenhoff, M. Scherf, M. Seifert, T. Borodina, A. Soldatov, D. Parkhomchuk, D. Schmidt, S. O'Keeffe, S. Haas, M. Vingron, H. Lehrach, and M. Yaspo. A Global View of Gene Activity and Alternative Splicing by Deep Sequencing of the Human Transcriptome. *Science*, 321(5891):956–960, 2008. 2, 19

[87] A. S. Tanenbaum. *Computer Networks (International Edition)*. Prentice Hall, fourth edition, August 2002. 35

[88] Life Technologies. Applied Biosystems. `http://www.appliedbiosystems.com/`, Janeiro 2012. 19

[89] W. Tolone, G. Ahn, T. Pai, and S. Hong. Access control in collaborative systems. *ACM Comput. Surv.*, 37(1):29–41, 2005. 35

[90] C. Trapnell, L. Pachter, and S. L. Salzberg. TopHat: discovering splice junctions with RNA-Seq. *Bioinformatics*, 25(9):1105–1111, 2009. 22

[91] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proceedings of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, Middleware '98, pages 55–70, London, UK, 1998. Springer-Verlag. 41

[92] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, Janeiro 2009. viii, 6, 7, 8

[93] D. Wall, P. Kudtarkar, V. Fusaro, R. Pivovarov, P. Patil, and P. Tonellato. Cloud computing for comparative genomics. *BMC Bioinformatics*, 11(1):259, 2010. 2, 26

[94] J. Wang, D. Crawl, and I. Altintas. Kepler + Hadoop: a general architecture facilitating data-intensive applications in scientific workflow systems. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science*, WORKS '09, pages 12:1–12:8, Nova York, NY, EUA, 2009. ACM. 26

[95] L. Wu and R. Buyya. Service Level Agreement (SLA) in Utility Computing Systems. Technical Report CLOUDS-TR-2010-5, Cloud Computing and Distributed Systems Laboratory, The University of Melbourne, Setembro 2010. 35

[96] L. Youseff, M. Butrico, and D. Da Silva. Toward a Unified Ontology of Cloud Computing. In *Proceedings of the Grid Computing Environments Workshop*, GCE '08, pages 1–10, Nov 2008. 7

[97] D. R. Zerbino and E. Birney. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research*, 18(5):821–829, 2008. 22

[98] L. Zhang, S. Gu, B. Wan, Y. Liu, and F. Azuaje. Gene set analysis in the cloud. *Bioinformatics*, 28(2):294–5, Janeiro 2012. 26