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

Arquitetura de um Controlador de SLA para Ambiente de Nuvens Federadas

Breno Rodrigues de Moura

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

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

Brasília
2017

Ficha catalográfica elaborada automaticamente,
com os dados fornecidos pelo(a) autor(a)

MM929a

Moura, Breno Rodrigues
Arquitetura de um Controlador de SLA para Ambiente de Nuvens Federadas / Breno Rodrigues Moura; orientador Aletéia Patrícia Favacho de Araújo. -- Brasília, 2017.
98 p.

Dissertação (Mestrado - Mestrado em Informática) - Universidade de Brasília, 2017.

1. Computação em Nuvem. 2. Federação de Nuvens. 3. Acordo de Nível de Serviço - SLA. 4. Controlador de SLA. I. Favacho de Araújo, Aletéia Patrícia, orient. II. Título.

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

Arquitetura de um Controlador de SLA para Ambiente de Nuvens Federadas

Breno Rodrigues de Moura

Dissertação apresentada como requisito parcial para
conclusão do Mestrado em Informática

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

Prof. Dr. José Viterbo Filho
IC/UFF

Prof.ª Dr.ª Maristela Terto de Holanda
CIC/UnB

Prof.ª Dr.ª Célia Ghedini Ralha
Coordenadora do Programa de Pós-graduação em Informática

Brasília, 23 de fevereiro de 2017

Dedicatória

Dedico este trabalho primeiramente a Deus, pois sem ele não somos nada, aos meus pais que se preocuparam comigo quando eu passava madrugadas no laboratório, que me apoiaram e me incentivaram a persistir e não desistir dos meus sonhos e trabalhos. Dedico a minha namorada/noiva, pela paciência e compreensão durante os dias e madrugadas as quais estava fazendo este trabalho, dedico também aos meus amigos, que aos quais sem esses não teria atingido os resultados obtidos no trabalho.

Agradecimentos

Agradeço aos professores e amigos, que me auxiliaram e motivaram para a realização deste trabalho. A minha orientadora, Aletéia Patrícia, pela paciência, dedicação e por nos motivar em seguir em frente com o trabalho nas horas de desespero e agonia por não conseguir avançar no trabalho. Agradeço a equipe BioNimbuZ, que por trabalhar em um mesmo ambiente, auxiliaram no desenvolvimento do mesmo, formando uma ótima equipe de trabalho. Pelas horas nas madrugadas e pelas risadas durante o desenvolvimento deste trabalho e pelos desesperos passados juntos, em especial meus amigos. Agradeço meus pais, que sempre me falaram para nunca desistir e persistir sempre, para que nosso objetivo fosse alcançado.

Resumo

Com o aumento de investimentos no setor da computação em nuvem e a disputa forte por consumidores que utilizem serviços em nuvens, ofertas de serviços surgem cada vez mais acirradas para conquistar o público consumidor deste mercado. E para continuar na disputa desse mercado, alguns provedores de nuvens se unem para formar federações. Esse processo envolve contratos, que visam garantir que esses provedores de serviços cumpram o que estão ofertando. Esses contratos denominados de Acordo de Nível de Serviço (em inglês, *Service Level Agreement - SLA*), são contratos que identificam as partes envolvidas em um negócio, além de especificar o mínimo de expectativas e limites que existe entre as partes, buscando melhorar a qualidade de serviço e a relação entre cliente e provedor. Assim, para melhor atender usuários de plataformas de nuvens federadas, este trabalho propõe a implementação de um controlador de SLA capaz de gerenciar os acordos de nível de serviço entre as nuvens federadas e os usuários de uma plataforma. Esse controlador deve trabalhar de forma dinâmica, automática, transparente e simples. Para comprovar a eficiência do controlador proposto foi utilizado como estudo de caso a plataforma BioNimbuZ para a implementação deste trabalho.

Palavras-chave: Computação em Nuvem, Federação de Nuvens, Acordo de Nível de Serviço - SLA, Controlador de SLA.

Abstract

With increasing investments in the cloud computing industry and the ever-growing batle for market share, cloud service offferings are becoming ever more fierce. In order to continue to dispute this market, some cloud providers have come together to form federations. This process involves contracts, which to ensure that these service providers comply with what they are offering. The contracts signed between the parties involved, the Service Level Agreement (SLA), is an IT service contract that specifies the minimum expectations and obligations that exist between the provider and the customer, aiming to improve the quality of service and the relationship between client and provider. Thus, to better meet the needs of users of federated cloud platforms, this work proposes the implementation of an SLA controller capable of managing service level agreements between federated clouds and users of a platform. This controller should work dynamically, automatically, transparently and simply. In order to prove the efficiency of the propose SLA controller the platform BioNimbuZ was used as a case study for the implementation of this work.

Keywords: Cloud Computing, Cloud Federation, Service Level Agreement - SLA, SLA Controller.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Em um cenário no qual a maior parte do uso computacional é realizado por servidores locais e *datacenters* proprietários, e o poder computacional nem sempre é utilizado no total da sua capacidade, paga-se por energia, manutenção e tecnologias que podem não ser usadas em sua totalidade. Neste contexto, surgiu a ideia de que, em vez de pagar por uma infraestrutura proprietária, de manutenção cara, que as vezes fica ociosa, fosse utilizado um ambiente em que o pagamento dos seus recursos fosse de acordo com a demanda de uso. Assim, nesse ambiente os recursos seriam alocados de acordo com a necessidade do usuário, resultando na computação sob demanda [56]. Nesse cenário, surgiu a plataforma de computação em nuvem, que busca reduzir o custo computacional, aumentar a confiabilidade e a flexibilidade para transformar o processamento dos dados e das aplicações em serviço, de maneira dinâmica e transparente [10].

Em sistemas de computação em nuvem diferentes características estão presentes, tais como a virtualização, a elasticidade, a escalabilidade, a interoperabilidade e os acordos de nível de serviço [86]. Essas características são necessárias para definir os serviços ofertados por provedores de nuvens, e para garantir que a entrega desses serviços seja feita de acordo com o que foi combinado com um provedor de nuvem, sem perda de desempenho, ou seja, sem perda da Qualidade de Serviço (*Quality of Service - QoS*) esperada pelo cliente.

Assim,nesse cenário é adequado firmar um Acordo de Nível de Serviço (em inglês, *Service Level Agreemente - SLA*), que são contratos firmados para identificar as partes envolvidas em um negócio. Além disso, esses contratos servem para especificar o mínimo de expectativas e limites que existe entre as partes envolvidas no négocio, buscando melhorar a qualidade de serviço e a relação entre cliente e provedor [8].

Dessa forma, para melhor atender os consumidores dos serviços de nuvem, em relação à necessidade de maior poder de processamento e maior capacidade de armazenamento, além da possibilidade de ter um custo reduzido para utilização desses recursos, surgiu a ideia de integrar nuvens. Essa integração de nuvens resultou na chamada federação de nuvens, que

são conjuntos de nuvens que possuem todos os seus recursos gerenciados por meio de uma interface conectada a todas elas, de forma que, se uma nuvem não tiver recursos para atender determinada demanda, uma outra nuvem, que esteja com recursos disponíveis no momento, possa ser integrada. Dessa forma, a integração feita entre diferentes nuvens possibilita a execução da demanda requisitada, de forma transparente ao consumidor [75].

Assim como na computação em nuvem, em federações de nuvens também é necessário um contrato entre as partes envolvidas, garantindo que esses provedores de serviços em nuvens cumpram o que estão ofertando. Todavia, em uma federação existem vários contratos, o contrato entre os provedores de nuvem e a plataforma de federação, e outro contrato entre a plataforma de federação e o consumidor do serviço em nuvem..Além disso, os trabalhos relacionados a implementação de um controlador de SLA, em sua maioria, não abordam todas as etapas do ciclo de vida do SLA, e a proposta que considera todas as etapas possui código fechado. Ainda nesse cenário, nenhum desses trabalhos considera características específicas das aplicações. Assim sendo, notou-se a necessidade de ter um único contrato que possa controlar a gerência desses acordos entre os provedores e o cliente final dos serviços [5].

Todavia, para que esse único contrato possa fazer a gerência dos serviços prestados, faz-se necessário estabelecer um controlador de SLA na plataforma de federação. Esse controlador deve, de forma transparente, contabilizar os cálculos de custos, gerenciar o ciclo de vida de cada contrato firmado, e monitorar os recursos contratados pelo consumidor [10].

Dessa forma, nota-se que cada plataforma de federação de nuvem deve ter um controlador de SLA, o qual deve ser capaz de garantir o cumprimento da Qualidade de Serviço contratada pelo consumidor final.

Nesse cenário, este trabalho propõe implementar um controlador de SLA dinâmico, transparente para o usuário e eficiente para o ambiente de nuvem federada. Assim, a plataforma de nuvem federada BioNimbuZ [72] foi escolhida como estudo de caso para a implementação do controlador de SLA. O BioNimbuz é uma plataforma para federação de nuvens híbridas, que foi proposta originalmente por Saldanha [74], e que tem sido aprimorada constantemente em outros trabalhos [3], [6], [56], [68], [72], [76]. Ele foi desenvolvido para suprir a demanda de plataformas de nuvens federadas, visto que a utilização de nuvens de forma isolada já não atende, em muito casos, às necessidades de processamento, de armazenamento, entre outras, na execução das aplicações de Bioinformática.

1.1 Motivação

A federação de nuvens tem se mostrado uma plataforma capaz de integrar diferentes infraestruturas, proporcionando uma maior flexibilidade e melhores preços na escolha de provedores/recursos.

Todavia, para que essa plataforma seja capaz de gerir diferentes contratos entre os provedores e o usuário da plataforma, garantindo a qualidade de serviço e o contrato independente do provedor usado, tem-se a necessidade de um controlador de SLA. Esse controlador deve ser o responsável por gerenciar os contratos firmados entre os usuários e a plataforma de nuvem federada. Além disso, ele deve contabilizar os cálculos de custos gerados pelo uso dos recursos alocados para as tarefas, gerenciar o ciclo de vida de cada contrato firmado, e monitorar os recursos contratados pelo consumidor, de forma transparente e automática.

Assim, diante dessa necessidade nas plataformas de nuvens federadas, a motivação deste trabalho é implementar um controlador de SLA, capaz de gerir os contratos entre diferentes provedores e o usuário, objetivando garantir uma arquitetura que atenda todas as etapas de um ciclo de vida de um SLA.

1.2 Problema

A negociação e o monitoramento de múltiplos SLAs em nuvens federadas necessita de colaboração entre os provedores de serviços para atender de forma satisfatória as requisições de serviços dos clientes. Isso torna-se uma tarefa muito complexa e desafiadora, pois a implementação de um controlador de SLA, em um ambiente de nuvens federadas, precisa fazer toda a gerência entre os provedores de nuvem. Essa gerência implica no monitoramento das escolhas dos recursos, da criação desses recursos, na execução das tarefas desses recursos, no monitoramento da execução das tarefas e no monitoramento do término dos recursos, controlando assim o ciclo de vida do contrato, e garantindo o custo, o tempo e as configurações desses recursos.

Contudo, essa tarefa tem sido desenvolvida em vários ambientes de forma manual. E isso dificulta para que a gestão da Qualidade de Serviço seja feita para uma grande quantidade de entidades envolvidas.

Nesse contexto, este trabalho propõe um controlador automatizado de SLA para nuvens federadas. Assim, com o controlador proposto é possível garantir o cumprimento dos acordos de níveis de serviços de maneira transparente e automática entre diversas nuvens.

1.3 Objetivos

1.3.1 Principal

O objetivo principal deste trabalho é desenvolver um controlador de SLA para plataformas de nuvens federadas, de forma dinâmica, automática, transparente e simples, utilizando como estudo de caso a plataforma de federação de nuvem BioNimbuZ.

1.3.2 Objetivos Específicos

Para cumprir o objetivo principal, este trabalho tem os seguintes objetivos específicos:

- Definir os níveis de serviço a serem controlados pelo SLA;
- Especificar a arquitetura do controlador de SLA para ambiente de nuvens federadas;
- Implementar o ciclo de vida do SLA;
- Integrar o controlador de SLA na plataforma BioNimbuZ;
- Realizar um estudo de caso, com acordos simulados no BioNimbuZ, para testar o desempenho e a qualidade do SLA na plataforma.

1.4 Estrutura do Trabalho

Este trabalho contém, além deste capítulo introdutório, mais seis capítulos. No Capítulo 2 são apresentados os conceitos de sistemas distribuídos e as definições de *clusters*, *grids* e *cloud*. Também são mostradas as diferenças entre os dois primeiros sistemas e a computação em nuvem, de forma a tornar claro esses conceitos.

O Capítulo 3 apresenta a plataforma BioNimbuZ, mostrando sua estrutura, sua organização, seus objetivos e as modificações sofridas desde a sua proposta original.

No Capítulo 4 são mostradas as definições de SLA, assim como seus componentes, sua estrutura e seus ciclos de vida. Além de abordar os principais trabalhos relacionados ao tema.

No Capítulo 5 é descrito o controlador de SLA proposto neste trabalho. Para isso, são apresentadas suas funcionalidades, sua integração com a plataforma BioNimbuZ e a sua implementação do ciclo de vida do SLA.

No Capítulo 6 são apresentados os resultados obtidos, com a implementação e a análise do estudo de caso do controlador de SLA na plataforma de nuvens federadas BioNimbuZ.

Para finalizar, no Capítulo 7 são apresentadas as considerações finais obtidas com a realização desta pesquisa, assim como alguns trabalhos futuros.

Capítulo 2

Computação em Nuvem

Neste capítulo serão apresentados alguns sistemas distribuídos, tais como o *cluster*, o *grid* e a computação em nuvem. Assim, nas Seções 2.1 e 2.2 serão apresentadas suas principais características, as suas arquiteturas, o modo como funcionam e as suas diferenças.

Além disso, a Seção 2.3 aborda as nuvens federadas que surgiram com a necessidade de aumentar o poder computacional e melhorar a provisão de serviços das nuvens. Nessa seção serão apresentados também os desafios de projeto relacionados à construção de uma federação de nuvens. Por último, a Seção 2.4 faz um resumo geral sobre os assuntos abordados neste capítulo.

2.1 Sistemas Distribuídos

Com a utilização cada vez maior dos computadores para realizar cálculos complexos, os computadores monoprocessados deixaram de ser suficientes e não estavam mais atendendo às necessidades. Assim, a solução usada por muitos anos foi uma escolha entre três possibilidades: aumentar a frequência do processador, melhorar o algoritmo e utilizar mais computadores [80].

A primeira solução foi bastante eficiente durante muitos anos, inclusive seguindo a Lei de Moore [54]. Esta solução, porém, esbarra em um limite físico e pode chegar ao fim a qualquer instante [48], por isso novas alternativas tem sido pesquisadas. A segunda solução pode não ser muito eficaz se tratando de alguns problemas, como cálculos de Bioinformática, previsão de clima e simulações de movimentação dos oceanos, que mesmo com algoritmos mais eficientes não podem ser processadas por uma máquina monoprocessada devido a sua complexidade. Isto nos leva à terceira solução, que consiste na utilização de mais de um computador, dando origem aos sistemas distribuídos.

O conceito de sistemas distribuídos já existe há algum tempo e possui diversas definições na literatura. Algumas definições clássicas são apresentadas a seguir:

- Para Lages e Nogueira [42], um sistema distribuído é "um conjunto de elementos de computação que cooperam, trocando informação";
- Pitanga [64] diz que "um sistema distribuído é um conjunto de elementos que se comunicam através de uma rede de interconexão e que utilizam software de sistema distribuído. Cada elemento é composto por um ou mais processadores e uma ou mais memórias";
- Colouris *et al.* [14] afirmam que "um sistema distribuído é aquele no qual os componentes localizados em computadores interligados se comunicam e coordenam suas ações apenas passando mensagens";
- Segundo Steem e Tanenbaum [82], um sistema distribuído é "uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente".

Pelas definições é possível identificar alguns aspectos relevantes que caracterizam um sistema distribuído, pois eles são formados por múltiplos computadores (unidades de processamento), estão interconectados por uma rede e se comunicam por meio de troca de mensagem e, portanto, não compartilham memória [42].

Contudo, projetar um sistema distribuído esbarra em algumas dificuldades, como a ausência de memória global, a ausência de um *clock* global e a imprevisibilidade no retardo das mensagens. Como não existe uma memória única, a comunicação ocorre por trocas de mensagem, ficando sujeita à imprevisibilidade no retardo das mesmas, ou seja, é difícil definir se uma determinada mensagem foi perdida ou apenas sofreu um atraso. E a falta de um *clock* global dificulta o ordenamento de eventos, um entrave para o estabelecimento de dependências entre ações [82].

Dessa forma, alguns aspectos devem ser considerados durante o projeto de um sistema distribuído, tais como transparência, tratamento de falhas, heterogeneidade, segurança, entre outros. As escolhas irão definir como funcionará o sistema, e quais tecnologias devem ou não ser utilizadas.

A transparência é muito importante para fazer com que os usuários tenham uma visão única do sistema. Ela pode ser dividida em dois níveis, o nível de usuário e o de programador. O usuário deve ter a impressão de que está utilizando um sistema monoprocessado, enquanto que o programador deve ter a impressão de estar programando em um sistema monoprocessado.

Outro aspecto relevante nos sistemas distribuídos é a heterogeneidade, que ocorre principalmente entre a rede, o hardware e o sistema operacional das máquinas. As diferenças na rede, geralmente, são resolvidas com a utilização de um protocolo, que consiste em uma convenção que permite a comunicação e a troca de dados. A heterogeneidade de

sistema operacional deve ser resolvida com a utilização de uma camada de software de alto nível, a qual é chamada de *middleware* [82].

O tratamento de falhas ocorre de diversas formas, começando pela detecção das falhas, passando pelo mascaramento das mesmas, até a recuperação. Nesse caso, a redundância é algo fundamental, visto que existem múltiplos computadores e esse potencial deve ser utilizado.

Além das características citadas, a segurança é fundamental em sistemas distribuídos, pois é uma das principais características visadas pelos consumidores de serviços em nuvem. A seguir serão mostrados alguns dos exemplos de sistemas distribuídos existentes, e serão feitas algumas comparações para diferenciá-los.

2.1.1 *Cluster*

A primeira iniciativa na construção *Clusters* Computacionais foi realizada pela IBM, em meados dos anos 60, como alternativa de interligar grandes *mainframes* para prover aos seus usuários uma forma comercial mais eficiente de paralelismo. Contudo, o conceito de clusterização não ascendeu como previsto até o surgimento de outras três tecnologias: microprocessadores de alta-performance, redes de alta velocidade e protocolos para computação distribuída de alta-performance [83].

Os recentes avanços dessas três tecnologias, somadas à sua acessibilidade (baixo preço decorrente da alta demanda) facilitaram a implementação de *clusters* computacionais como solução do antigo problema da eficiência de custos na busca de sistemas paralelos de alta performance. Contudo, para se construir um *cluster*, os computadores componentes devem ter alguns elementos essenciais [9]:

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

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

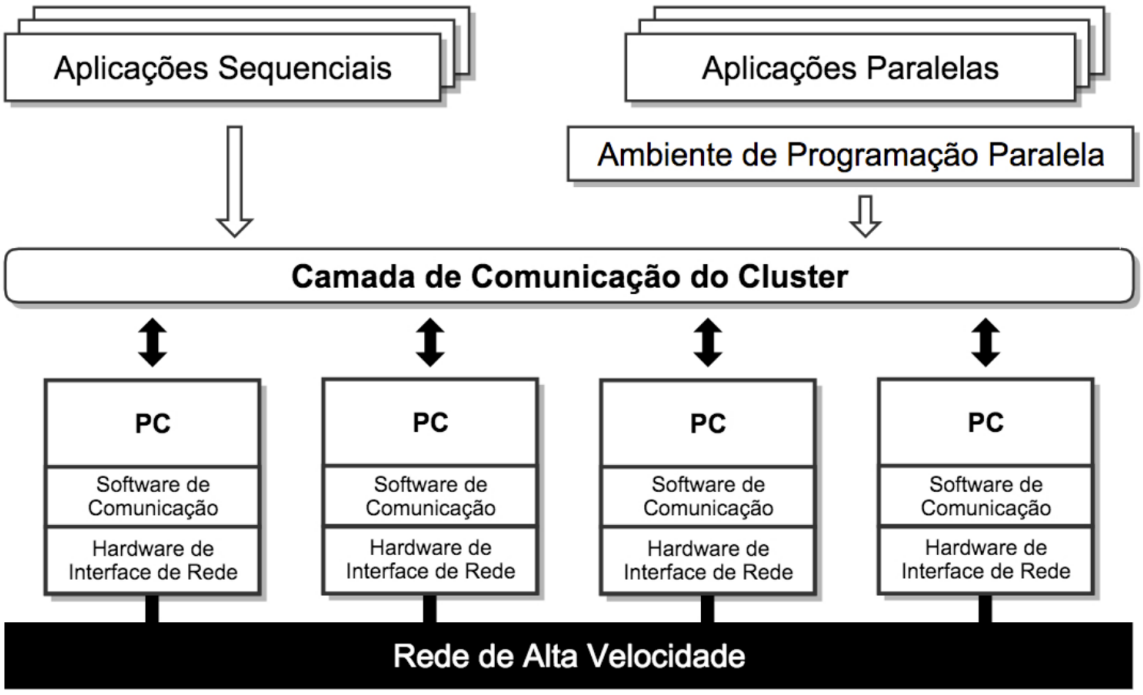

Figura 2.1: Arquitetura de um *Cluster*, adaptada de [9].

Na busca para provar o ganho de performance de *clusters* sobre as plataformas tradicionais de sistemas distribuídos, diversos projetos acadêmicos surgiram, tais como o Beowulf [83], o Berkeley NOW [92] e o HPVM [51].

Beowulf foi um *cluster* construído em 1994 por Thomas Sterling e Don Becker e consistiam de 16 processadores DX4 conectados por um canal Ethernet, dedicados à programação paralela [9]. O *cluster* criado foi muito bem visto pela área acadêmica e comercial, tanto que a NASA se interessou por este novo modelo. Em pouco tempo, o *cluster* Beowulf foi bastante difundido, tornando-se "Projeto Bewoulf" e passou a ser visto como gênero dentro da comunidade de Computação de Alta Performance (*High Performance Computing*).

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

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

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

2.1.2 *Grid* Computacional

Em meados dos anos 90 o termo *Grid* Computacional foi criado para descrever tecnologias que possibilitariam usuários obterem poder computacional sobre demanda. Este termo é uma analogia com as redes de energia elétrica (do inglês, *power grids*), pois é um sistema que está presente em vários lugares, e que é acessado pelas pessoas de forma simples e natural, sem a preocupação de como aquela energia foi produzida [7].

Para Buyya *et al.* [11], um *grid* é um tipo de sistema distribuído e paralelo que permite o compartilhamento, a seleção e a agregação de recursos geograficamente distribuídos em tempo de execução e de forma dinâmica. Por esta definição, percebe-se que *grid* computacional é um sistema distribuído que utiliza recursos que estão dispersos geograficamente ao longo do planeta. Esses recursos são interligados por uma rede de alta velocidade e, na maioria das vezes, são altamente heterogêneos, ou seja, compreendem computadores com configurações, hardware e software diferentes, que são combinados para aumentar o poder computacional do sistema como um todo.

Este paradigma de computação evoluiu bastante no ramo científico, e projetos utilizando este tipo de sistema distribuído surgiram, como por exemplo o *Open Science Grid* [24], que possui estudos em diversas áreas, entre elas a física, as nanopartículas e a biologia molecular.

Apesar de ser bastante utilizado no meio científico, é complexo desenvolver aplicações que executem neste tipo de plataforma, pois diversas questões devem ser levadas em consideração, como a heterogeneidade, o dinamismo, a escalabilidade e a segurança. Como existem diversos computadores conectados, deve-se levar em conta as possíveis diferenças entre hardware e software. Quanto ao dinamismo e escalabilidade, deve-se estar atento quanto à disponibilidade, ou seja, um computador pode não estar mais disponível para o *grid*, caso esteja sendo utilizado pelo usuário. E garantir a segurança é um problema, pois os recursos estão espalhados em diversos lugares e o controle sobre cada um deles se torna complicado.

Para tentar superar estas dificuldades, Foster *et al.* [21] propuseram uma arquitetura organizada em camadas, e que é apresentada na Figura 2.2.

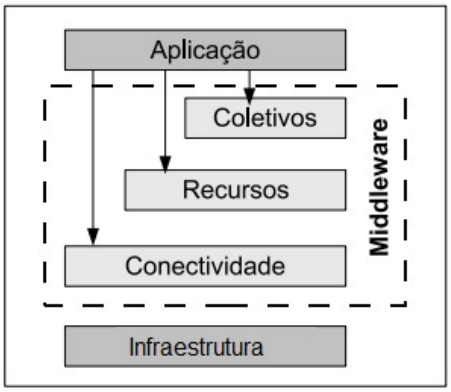

Figura 2.2: Camadas da Arquitetura Grid [21].

A primeira camada, de baixo para cima é a de infraestrutura, nela se encontram os recursos computacionais propriamente ditos, que realizam o processamento e o armazenamento, entre outros. No meio existe o *midleware*, uma camada de software responsável por prover transparência ao usuário e facilitar a utilização deste ambiente. Nesta arquitetura, tem-se a divisão do *midleware* em três camadas distintas. Na camada de conectividade se encontram o núcleo de comunicação e os protocolos para transmissão em redes. É por meio dela que é possível estabelecer uma comunicação entre os diversos recursos heterogêneos presentes no *grid*. A camada de recursos tem a responsabilidade de inicializar e controlar o compartilhamento de recursos individuais. A camada coletivos abrange aquilo que não pertence a algum serviço específico, como escalonamento, monitoramento e replicação de dados. E por fim, a camada de aplicação deve fornecer facilidades para que as aplicações possam ser executadas, sem que seja necessário conhecer especificamente cada componente do *grid* [21].

Nesse cenário, é comum haver confusão nas definições das plataformas de *grids* e de nuvem, pois ambas possuem algumas características em comum, apesar de terem focos diferentes. A Seção 2.2.4 explicará um pouco mais sobre estas diferenças, mas antes será apresentado o conceito de nuvem computacional.

2.2 Nuvem Computacional

A computação em nuvem é um paradigma de computação distribuída que surgiu como uma tendência para a provisão de recursos e serviços de forma rápida, barata, escalável e flexível [22]. Diversos tipos de recursos podem ser disponibilizados, tais como memória, capacidade de processamento, de armazenamento, entre outros. Na literatura há várias

definições para o ambiente de nuvem computacional, algumas dessas definições serão descritas a seguir:

- Armbrust *et al.* [2] definem como "a união de aplicações oferecidas como serviço pela Internet, com o hardware e o software localizados em *datacenters* de onde o serviço é provido";
- De acordo com Foster *et al.* [22] computação em nuvem é "um paradigma computacional altamente distribuído, direcionado por uma economia de escala, na qual poder computacional, armazenamento, serviços e plataformas abstratas, virtualizadas, gerenciadas e dinamicamente escaláveis são oferecidos sob demanda para usuários externos por meio da Internet";
- Buyya *et al.* [11] definem como "um tipo de sistema paralelo e distribuído, que consiste em uma coleção de computadores virtuais interconectados que são provisionados dinamicamentes e apresentados como um ou mais recursos computacionais unificados, baseados em acordos de nível de serviço estabelecidos entre provedor de recursos e o consumidor".

A primeira definição é bastante abrangente e deixa espaço para que haja confusão com outros paradigmas de computação distribuída. É possível, por exemplo, confundir com a definição de um *grid* computacional. A segunda já apresenta um elemento importante para a definição mais completa de nuvem, que é a frase "oferecido sob demanda". Isto significa que na computação em nuvem o usuário paga apenas pelo o que foi consumido, diferentemente de como ocorre nos outros paradigmas de computação distribuída, e isso é uma diferença fundamental. A terceira definição fala sobre os acordos de nível de serviço. Esse acordo é negociado entre o consumidor e o fornecedor, e é ele quem define quais indicadores irão medir a qualidade do serviço. Assim, a violação deste acordo pode levar ao pagamento de multas [11].

Dessa forma, a computação em nuvem possui diversas características que a difere dos outros sistemas distribuídos existentes. As principais características são [50]:

- *Self-service*: na computação em nuvem, os serviços, tais como armazenamento e processamento, são contratados pelo usuário diretamente, sem a participação de algum administrador dos provedores de nuvem, e de forma que atenda às necessidades do usuário [86];
- Amplo Acesso: a ideia é que os recursos estejam disponíveis na Internet e possam ser acessados a partir de dispositivos heterogêneos, desde que estes possuam acesso à rede mundial de computadores;

- *Pooling* de Recursos: os recursos de um determinado provedor são organizados em um *pool* de recursos físicos e virtuais, de forma a facilitar o acesso de vários usuários e facilitar também os ajustes de demanda;
- Elasticidade: caso seja necessário aumentar a utilização de determinado recurso, seja ele qual for, este aumento deve ocorrer de forma fácil ou até mesmo de forma automática;
- Serviço Medido: a utilização dos recursos deve ser monitorada a todo momento, de forma a garantir que seja cumprido o contrato de qualidade de serviço realizado entre o provedor de serviço e o usuário.

2.2.1 Arquitetura de uma Nuvem

Na literatura há diferentes propostas de arquitetura de nuvens computacionais [44] [91], e a Figura 2.3 apresenta um destes modelos. Nela é possível identificar três atores, que são os Provedores de Serviço (*Service Providers*), os Provedores de Infraestrutura (*Infrastructure Providers*) e os Usuários de Serviço (*Service Users*).

Os provedores de infraestrutura disponibilizam recursos para que os provedores de serviços hospedem suas aplicações sem precisar se preocupar com as questões de estrutura física, a fim de ganhar escalabilidade e flexibilidade. Os usuários de serviço acessam as aplicações que foram colocadas a disposição através da Internet. Tanto os usuários de serviço quanto os provedores de serviço pagam apenas por aquilo que consumirem, o chamado *pay-per-use* [44].

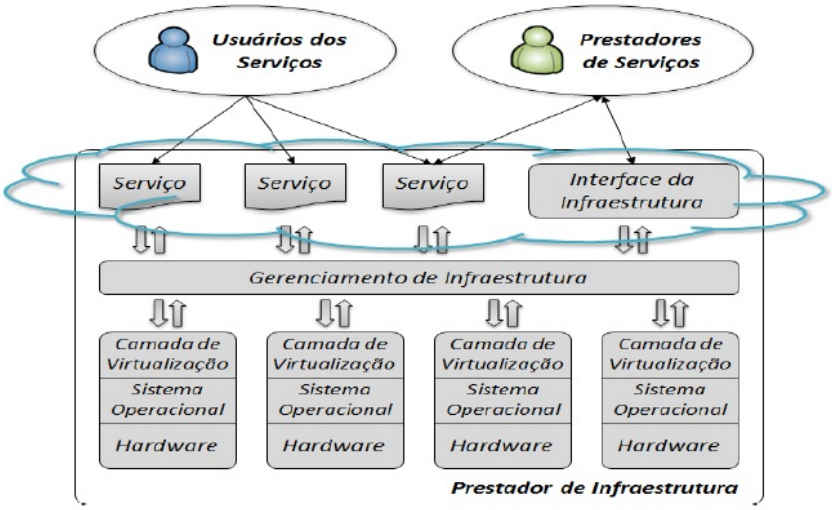

Figura 2.3: Arquitetura de Nuvem Computacional, Proposta por Vaquero *et al.* [86].

Esta arquitetura de nuvem está dividida em três camadas distintas. A mais próxima do usuário é a que se encontram os serviços. Assim, ela possui uma interface para que os clientes tenham acesso às aplicações que foram disponibilizadas. A camada mais baixa é a infraestrutura de fato. Ali estão localizados os servidores, os *datacenters*, a rede e toda a parte física que compõe a nuvem. A camada do meio fornece facilidades para que o provedor de serviços consiga disponibilizar suas aplicações sem ter que se preocupar com as individualidades do hardware.

Como não há um padrão quanto a arquitetura de uma plataforma de nuvem, este trabalho apresenta também a proposta de Foster *et al.* [22] que está representada na Figura 2.4.

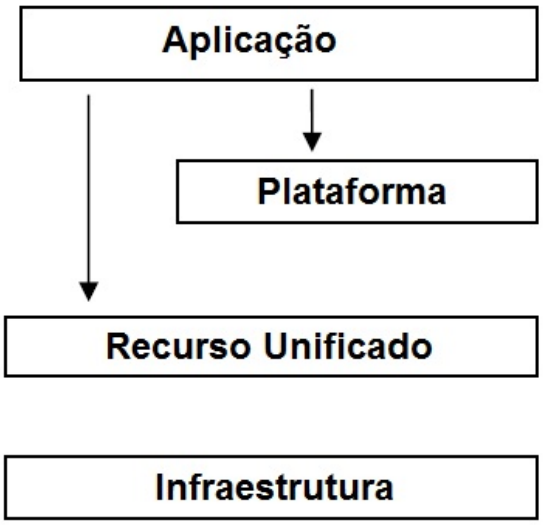

Figura 2.4: Arquitetura para a Plataforma de Nuvem, Proposta por Foster *et al.* [22].

Nessa arquitetura a camada de infraestrutura contém os recursos computacionais, como os recursos de armazenamento e os de processamento, que representam o hardware propriamente dito. A camada de recursos unificados contém os recursos que foram encapsulados, geralmente, por meio da virtualização, e estão disponíveis tanto para os usuários finais quanto para a camada superior. Assim, *clusters* e computadores virtuais são exemplos de recursos desta camada. A camada de plataforma consiste em um conjunto de ferramentas especializadas que estão executando em cima da camada de recursos unificados, tais como plataformas de desenvolvimento. E por fim, a camada de aplicação que contém as aplicações que executam na nuvem [22].

2.2.2 Modelos de Serviços

Os serviços que são oferecidos no paradigma de computação em nuvem podem ser divididos em categorias. Apesar de ser possível encontrar na literatura propostas com mais de três classes [70], o mais comum é dividir os serviços em *Infrastructure-as-a-Service*, *Platform-as-a-Service* e *Software-as-a-Service*, descritos a seguir:

- *Infrastructure-as-a-Service*: os serviços que são oferecidos se caracterizam por serem de infraestrutura, podendo ser desde capacidade de processamento, do armazenamento ou até máquinas virtuais completas. Um provedor desta camada que se destaca é a Amazon, com o *Elastic Compute Cloud* (EC2) [45] e com o *Simple Storage Service* (S3) [46];
- *Platform-as-a-Service*: o que caracteriza este nível é a disponibilização de um ambiente no qual o usuário possa programar, testar e executar aplicações. O Google App Engine [30] se encontra nesta categoria, pois é um ambiente no qual aplicações podem ser desenvolvidas sem ter nenhum tipo de programa instalado na sua máquina, tudo é feito através da Internet;
- *Software-as-a-Service*: são as aplicações disponibilizadas pelos provedores como serviços aos usuários comuns, de forma gratuita ou cobrando pela sua utilização. Um exemplo é o Google Docs [32], no qual são disponibilizados editores de textos, editores de planilhas e ferramentas para a criação de apresentações. Os usuários podem acessar estes serviços em qualquer lugar, sem a limitação de ter o software instalado na sua máquina.

2.2.3 Tipos de Nuvem

As nuvens podem ser divididas em quatro tipos diferentes de implantação, que são nuvens públicas, privadas, comunitárias e híbridas [4]:

- Nuvem Pública: esse tipo de nuvem é aquela mantida por um fornecedor, geralmente uma grande companhia, para um usuário comum ou uma empresa. É a nuvem no sentido tradicional do senso comum, na qual os recursos são provisionados dinamicamente e por meio da Internet *web-services* e aplicações web são disponibilizadas [70];
- Nuvem Privada: é a nuvem que está dentro de um contexto empresarial e não está acessível a todas as pessoas, sendo restrita apenas aos funcionários e aos parceiros da empresa. Pelo fato de não estar disponível na Internet, apresenta vantagens em termos de segurança e de largura de banda da rede;

- Nuvem Comunitária: infraestrutura que é compartilhada por organizações que mantém algum tipo de interesse em comum (jurisdição, segurança, economia), e pode ser administrada, gerenciada e operada por uma ou mais destas organizações;
- Nuvem Híbrida: ambiente no qual ambos os tipos de nuvens anteriormente descritos podem ser utilizados em conjunto, em que nuvens privadas, comunitárias e públicas podem interagir entre elas, compartilhando recurso de forma transparente para o usuário.

2.2.4 Comparação entre Sistemas Distribuídos

Diante das definições apresentadas nas seções anteriores, esta seção aborda a diferença entre as três plataformas: *cluster*, *grid* e nuvem. Algumas características fundamentais para a compreensão das diferenças entre as plataformas *cluster*, *grid* e nuvem são apresentadas na Tabela 2.1.

Tabela 2.1: Características de *Clusters*, *Grids* e Nuvens.

Característica	**Cluster**	**Grid**	**Nuvens**
`Recursos e Configurações Heterogêneas`	Não/Pouco	Sim	Sim
`Dedicado`	Sim	Não	Sim/Não
`Proximidade dos Componentes`	Sim	Não	Não
`Componentes` *Commodity*	Sim	Sim	Sim

Na Tabela reftab:diferenca o primeiro item é sobre a heterogeneidade, que nos *clusters* está pouco ou não está presente, pois eles são formados por componentes de hardware semelhantes e, geralmente, todos os nós executam o mesmo sistema operacional, enquanto que os *grids* e as nuvens são altamente heterogêneos, tanto no hardware quanto no software [22].

Em relação à dedicação, o *grid* não é dedicado, isto é, ele utiliza o poder de processamento dos computadores conectados quando estes não estão sendo utilizados pelos seus proprietários, enquanto que um *cluster* necessita de exclusividade para realizar o processamento e a nuvem pode ser ou não dedicada, pois é possível prover recursos dedicados como recursos virtuais. Além disso, os *clusters* utilizam componentes que estão muito próximos um do outro, geralmente, estão na mesma rede para realizar comunicação entre os mesmos, enquanto que os *grids* e as nuvens podem utilizar componentes que estão espalhados pelo planeta. Ambos possuem componentes *commodity*, isto é, componentes que auxiliam no desenvolvimento das tecnologias, como linguagens, APIs, SDKs, protocolos, sistemas operacionais e metodologias, são facilmente adquiridos a um custo baixo [7] [42].

Além dessas características, Ian Foster*et al.* [22] apresentaram algumas outras características e mostraram as diferenças em cada plataforma distríbuida:

- Modelo de Negócio: tradicionalmente os usuários estão acostumados a pagar uma vez por uso ilimitado de um determinado software. Com a computação em nuvem isto não é mais verdade, e é possível pagar apenas por aquilo que é consumido, assim como é feito com a água e a eletricidade. Porém, no caso da computação em nuvem, o pagamento pode ser feito por horas de utilização ou por quantidade de espaço utilizado. Já os *grids*, que são utilizados para fins científicos, utiliza um modelo baseado em projetos, que tenta prever o quanto de recursos será utilizado em um determinado projeto;

- Modelo de Computação: a computação em nuvem permite que várias aplicações sejam executadas ao mesmo tempo, graças a utilização da virtualização. No *grid* isto não é possível, apenas uma aplicação pode executar por vez;

- Virtualização: a virtualização é essencial para que exista a computação em nuvem, pois ela fornece a abstração e o encapsulamento necessários. Na nuvem é preciso que vários usuários utilizem os recursos sem perceber que existem outros usuários concorrendo, e cada um deles pode utilizar a quantidade de recursos que precisar. A virtualização fornece essa abstração sobre a infraestrutura. Já os *grids* não são dependentes de virtualização e não o utilizam na maioria das vezes, devido às políticas das organizações de ter controle total sobre os seus dados;

- Localização dos Dados: com o volume dos dados aumentando cada vez mais, deve-se considerar o custo de transmissão de um dado, do local onde está armazenado para o local no qual será processado. Existe uma grande diferença do custo de transmissão de um dado armazenado no próprio disco da máquina na qual será processado, do custo de transmissão pela rede. Quanto mais próximo estiver o servidor onde o dado está armazenado da máquina para o qual será levado, menor o custo de transmissão e, portanto, melhor o desempenho. É importante então levar em conta a localização física dos servidores no momento do armazenamento de um dado, e isto é comum na computação em nuvem. No *grid*, geralmente, é utilizado um sistema de arquivos distribuídos, como por exemplo, o NFS (*Network File System*) [60], o que dificulta a aplicação de decisões de armazenamento que levam em consideração a localidade.

Buyya *et al.* [11] fizeram uma comparação entre *cluster*, *grid* e nuvem, que pode ser vista na Tabela 2.2.

Contudo, apesar da computação em nuvem ser mais escalável que outros sistemas distribuídos, como os *clusters* e os *grids*, atualmente, ela já não é mais suficiente, em

Tabela 2.2: Diferenças entre os Sistemas Distribuídos, adaptado de Buya *et al.* [11].

Sistemas / Características	*Cluster*	*Grid*	Nuvem
Componentes	Computadores *commodities*	Computadores de alta tecnologia	Computadores *commodities* e de alta tecnologia
Sistema Operacional	Um SO padrão	Qualquer SO padrão	Máquinas virtuais com múltiplos SOs
Proprietário	Único	Múltiplo	Único
Rede de Conexão	Dedicada, largura de banda alta com baixa latência	Internet, alta latência e pouca largura de banda	Dedicada, largura de banda alta com baixa latência
Segurança	Tradicional, baseada em *login* e senha	Par de chave pública e privada com autenticação	Cada usuário utiliza uma máquina virtual, com suporte para controle de acesso.
Preço dos Serviços	Limitado, não aberto ao mercado	Dominado pelo interesse público ou privado	Preços utilitários com descontos para grandes clientes
Negociação de Serviço	Limitada	Sim, baseada em acordos de níveis de serviço	Sim, baseada em acordos de níveis de serviço
Gerenciamento de Usuário	Centralizado	Descentralizado	Centralizado ou pode ser delegado a um terceiro
Gerenciamento de Recursos	Centralizado	Distribuído	Centralizado/ Distribuído
Alocação/ Escalonamento	Centralizado	Descentralizado	Centralizado/ Descentralizado

alguns casos, de suprir sozinha a demanda por recursos computacionais. A solução então foi integrar mais de uma nuvem computacional, o qual foi chamado de federação de nuvens, que será descrita em detalhes na próxima seção.

2.3 Federação de Nuvens

Com o passar dos anos, novas necessidades foram surgindo e a utilização de nuvens de forma isolada passou a não ser mais suficiente para algumas aplicações. Além disso, empresas começaram a criar *datacenters* ao redor do mundo para aumentar a segurança em caso de queda em algum dos servidores, e também como uma forma de atender às solicitações em menor tempo, diminuindo a distância entre o usuário e os servidores. Dessa

forma, integrar nuvens passou a ser algo necessário para continuar fornecendo serviços de forma rápida, eficiente e escalável. Assim sendo, a federação de nuvens computacionais pode ser definida como um conjunto de provedores de nuvens públicos e privados, conectados através da Internet [75].

Bittman [13] dividiu a evolução do paradigma de computação em nuvem em três fases. A primeira fase é chamada de monolítica e se caracteriza pelas ilhas proprietárias, com serviços fornecidos por empresas de grande porte, como Google [31], Amazon [45] e Microsoft [53]. Na segunda fase, chamada de cadeia vertical de fornecimento, ainda se tem o foco nos ambientes proprietários, mas as empresas começam a utilizar alguns serviços de outras nuvens, sendo vista como o começo da integração. E por último, tem-se a federação horizontal, na qual pequenos provedores se juntam para aumentar sua escalabilidade e eficiência na utilização dos recursos, e começam a ser discutidos padrões de interoperabilidade.

Atualmente, tem-se vivido a terceira fase, isto é, a etapa em que os provedores de nuvem trabalham exclusivamente sozinhos está ficando para trás, e tem-se alcançado a etapa em que os pequenos provedores se aliam horizontalmente. Realizar uma federação, no entanto, não é algo simples devido as diferenças entre as nuvens, cada uma com suas particularidades, tanto no hardware (arquitetura do processador, por exemplo) quanto no software (sistema operacional ou outros softwares utilizados). Contudo, para que haja a federação é necessário que os seguintes requisitos sejam atendidos [12]:

- Automatismo e Escalabilidade: uma nuvem, utilizando mecanismos de descoberta, deve ser capaz de escolher, dentre as nuvens existentes, qual que atende às suas necessidades e deve ser capaz de reagir a mudanças;
- Segurança Interoperável: é necessário a integração entre diversas tecnologias de segurança, de forma que uma nuvem não precise alterar suas políticas de segurança para se juntar à federação.

Outros desafios para a criação de nuvens federadas foram identificados e são descritos a seguir [10]:

- Previsão de Comportamento da Aplicação: é importante que o sistema seja capaz de prever o comportamento das aplicações, para que ele possa tomar decisões inteligentes quanto à alocação de recursos, dimensionamento dinâmico, armazenamento ou largura de banda. Um modelo deve ser construído para tentar realizar esta previsão. Este modelo deve levar em consideração estatísticas de padrões de utilização dos serviços e ajustar as variáveis sempre que for necessário, para que o modelo seja o mais próximo possível da realidade;

- Mapeamento Flexível de Recursos e Serviços: algo que sempre é levado em consideração em qualquer projeto é o custo. É importante que o sistema seja o mais eficiente possível, sempre tentando encontrar a melhor configuração de hardware e software que atenda às demandas de qualidade de serviço que foram estabelecidas entre o usuário e o provedor. Esta é uma tarefa bastante complicada devido ao comportamento não previsível das aplicações e serviços que são utilizados na nuvem;
- Modelo Econômico Impulsionado por Técnicas de Otimização: o problema da tomada de decisões orientadas ao mercado é um problema de otimização combinatória, que visa encontrar a melhor combinação entre serviços e planos. Os modelos de otimização visam otimizar tanto os recursos centralizados (utilização, disponibilidade e incentivo) quanto os centrados nos usuários (tempo de resposta, o orçamento gasto e a justiça);
- Integração e Interoperabilidade: muitas empresas possuem um conjuntos de programas legados, de forma que a migração desses se tornam inviável, por fatores de segurança e de disponibilidade. Além disso, realizar a interoperabilidade de sistemas legados por meio da Internet, torna-se muito complexo e oneroso, devido a forma em que essas aplicações interagem com os dados e as aplicações que estão na nuvem;
- Monitoramento Escalável dos Componentes do Sistema: atualmente as técnicas que são utilizadas para o monitoramento e o gerenciamento dos componentes da nuvem utilizam uma abordagem centralizada. Em sistemas distribuídos manter uma singularidade sempre é um problema, principalmente, por este poder se tornar um gargalo para o sistema, caso o volume de requisições seja muito alto, como também por questões de segurança, visto que algum problema pode prejudicar toda a federação de nuvens. É importante que este monitoramento seja feito de forma distribuída, para que não haja problema de escalabilidade, de desempenho e de confiabilidade.

2.3.1 Arquitetura de uma Federação

Para que os requisitos anteriormente citados sejam atendidos e os desafios superados, arquiteturas para a federação de nuvens foram propostas [10] [12]. Celesti *et al.* [12] propuseram uma arquitetura para federação em nuvem, veja a Figura 2.5. Neste modelo eles dividiram as nuvens em dois grupos, local e externa. A nuvem local consiste naquele provedor que já atingiu a saturação, não consegue mais atender às demandas e, portanto, deve repassar requisições para o restante da federação. Os provedores que vão receber essas requisições são chamados de externos, e podem ou não cobrar por ceder seus recursos ociosos para atender às demandas que foram repassadas a ele.

Vale ressaltar que uma nuvem pode ser local e externa ao mesmo tempo, basta que ela esteja saturada em um determinado recurso, e por isso precise da ajuda de outros provedores para suprir essa demanda, e ao mesmo tempo está com outro recurso ocioso e o disponibiliza para outro provedor. Este repasse de requisições de uma nuvem para outra deve ser feito de forma transparente ao usuário, e de forma a atender os contratos de serviço que foram efetuados previamente.

Na arquitetura proposta por Celesti *et al.* [12], para cada provedor existente na federação existe um gerenciador chamado de *Cross-Cloud Federation Manager* (CCFM). O CCFM é o responsável por realizar a gerência das nuvens, verificando quando uma nuvem já está saturada e tentando encontrar nuvens que estejam dispostas a contribuir com seus recursos ociosos. Ele está dividido em três subcomponentes, cada um responsável por realizar uma fase dentre as três que dão nome a arquitetura, os quais são:

- *Discovery Agent*: é o agente responsável por realizar o processo de descoberta na federação. Para que este processo seja feito de forma dinâmica e distribuída, é necessário que cada provedor de nuvem disponibilize suas informações para um local centralizado (apesar de ser logicamente centralizado é implementado de forma distribuída), e sempre que uma nuvem precisar de informações sobre os outros provedores basta consultar este local;
- *Match-Making Agent*: este agente é o responsável por escolher dentre todas as nuvens estrangeiras, qual é a que melhor se encaixa nos requisitos pretendidos. Depois de feita a escolha, ele também é responsável por garantir que os acordos de níveis de serviço sejam cumpridos e a qualidade de serviço seja mantida;
- *Authentication Agent*: agente que tem como função realizar a autenticação dos usuários dentre as diversas nuvens da federação. É uma tarefa complicada, pois cada nuvem pode ter vários usuários autenticados, e essas autenticações podem variar a todo instante. Outro problema é que cada nuvem pode utilizar uma tecnologia de autenticação diferente da outra, e este agente é responsável por realizar a interoperabilidade entre essas tecnologias.

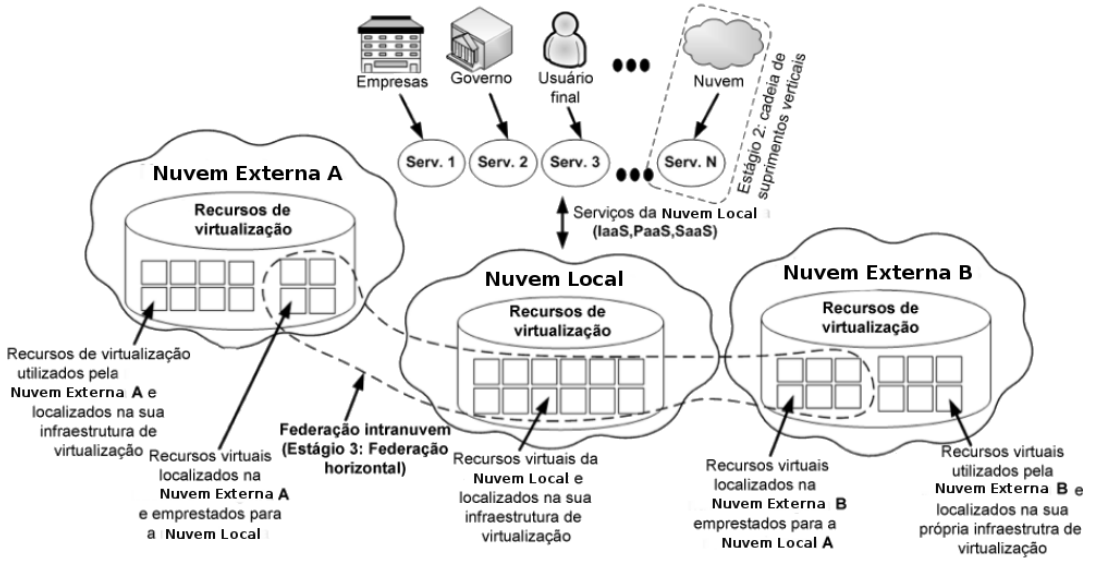

Figura 2.5: Arquitetura para Federação de Nuvens, proposta por Celesti *et al.* [12].

Uma proposta alternativa de arquitetura foi apresentada por Buyya *et al.* [10]. Nessa proposta, o usuário utiliza um componente externo aos provedores para realizar qualquer tipo de interação com a federação. Este componente é chamado de *Cloud Broker* (CB). Ele é o responsável por intermediar a comunicação entre o usuário e os provedores, e também por identificar os recursos que estão disponíveis em cada provedor, de forma a atender os requisitos de qualidade de serviço (QoS) que foram definidos através de um contrato de SLA.

Para conseguir estes dados sobre os provedores de nuvem, o CB consulta o *Cloud Exchange* (CEx), um outro componente da arquitetura. A principal função do CEx é servir como um registro, que é consultado pelo CB a fim de obter informações sobre a infraestrutura, o custo de utilização, os padrões de execução e os recursos disponíveis, além de mapear as requisições dos usuários aos provedores.

Em cada provedor presente na federação existe um outro componente, chamado *Cloud Coordinator* (CC), responsável por incluir a infraestrutura disponível na federação e expor estas informações aos interessados. Para atender às requisições dos usuários, o CC realiza uma autenticação e estabelece um acordo de qualidade de serviço com cada CB, que também é responsável por encaminhar as tarefas para a execução. Um exemplo desta arquitetura é apresentado na Figura 2.6.

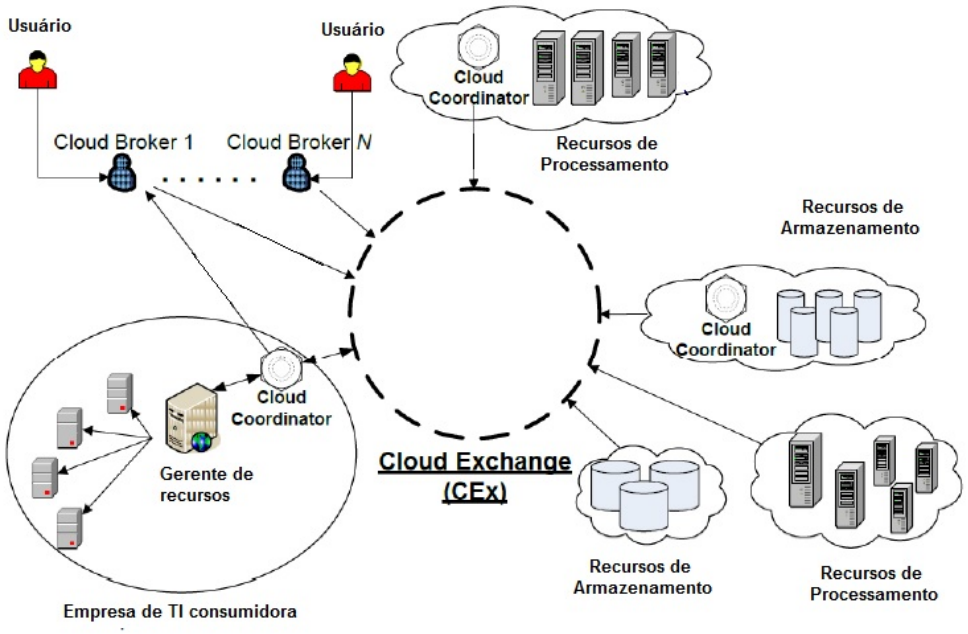

Figura 2.6: Arquitetura para Federação de Nuvens, proposta por Buyya *et al.* [10].

Como não existe um padrão para a federação de nuvens computacionais, uma nova arquitetura foi proposta, chamada BioNimbuZ [74], que tem como objetivo garantir de forma dinâmica, transparente e escalável a execução de aplicações no nível de *Software-as-a-Service*. Esta proposta é detalhada no próximo capítulo.

2.4 Considerações Finais

Neste capítulo foram descritos os conceitos de computação em nuvem, a sua arquitetura e os seus principais modelos de serviços. Além disso, foram apresentados os problemas encontrados no ambiente de nuvem por conta dos seus recursos as vezes não atenderem as necessidades de certas aplicações. Nesse cenário, discutiu-se também a plataforma de federação de nuvens, que busca implantar a ideia de que os recurso são infinitos.

Dentre os desafios e problemas encontrados em uma federação de nuvens, o acordo de nível de serviço é um problema em aberto, o qual será abordado no Capítulo 4, e para a implementação desse acordo em uma ambiente de nuvens federadas, faz-se necessário o uso de uma plataforma de federação. Assim, a plataforma BioNimbuZ foi escolhida para estudo de caso, deste trabalho porque a mesma possui código aberto e não possui um controlador de SLA. Logo, para melhor entender a sua arquitetura e as suas necessidades, o Capítulo 3 apresentará essas características, de forma a suprir a necessidade de um controlador de SLAs.

Capítulo 3

Plataforma BioNimbuZ

Este capítulo tem como objetivo apresentar a plataforma para federação de nuvens computacionais BioNimbuZ [56]. Para isso, serão detalhados seu funcionamento, sua arquitetura e os seus serviços e controladores. Além disso, serão apresentadas as modificações que ocorreram na proposta original, e quais tecnologias foram incorporadas com estas alterações.

3.1 Visão Geral

O BioNimbuZ é uma plataforma para federação de nuvens híbridas, que foi proposta originalmente por Saldanha [74], e que tem sido aprimorada constantemente em outros trabalhos [3], [6], [56], [68], [72], [76]. Ele foi desenvolvido para suprir a demanda de plataformas de nuvens federadas, visto que a utilização de nuvens de forma isolada já não atende, em muito casos, às necessidades de processamento, de armazenamento, entre outras, na execução das aplicações de Bioinformática.

O BioNimbuZ permite a integração entre nuvens de diversos tipos, tanto privadas quanto públicas, deixando com que cada provedor mantenha suas características e políticas internas, e oferece ao usuário a transparência e a ilusão de infinidade de recursos. Desta forma, o usuário pode usufruir de diversos serviços sem se preocupar com qual provedor está sendo de fato utilizado.

Outra característica desta plataforma é a flexibilidade na inclusão de novos provedores, pois são utilizados *plugins* de integração que se encarregam de mapear as requisições vindas da arquitetura para as requisições de cada provedor especificamente. Isso é fundamental para que alguns objetivos possam ser alcançados, principalmente, na questão da escalabilidade e da flexibilidade.

Originalmente, toda a comunicação existente no BioNimbuZ era realizada por meio de uma rede *Peer-to-Peer* (P2P) [84]. Porém, para alcançar os objetivos desejados de

escalabilidade e de flexibilidade, percebeu-se a necessidade de alterar a forma de comunicação entre os componentes da arquitetura do BioNimbuZ, pois a utilização de uma rede de comunicação P2P não estava mais suprindo as necessidades nestes dois quesitos. É importante ressaltar que os outros objetivos propostos por Saldanha [75], tais como obter uma arquitetura tolerante a falhas, com grande poder de processamento e de armazenamento, e que suportasse diversos provedores de infraestrutura foram mantidos e melhorados.

Assim sendo, no trabalho Moura *et al.* [56] implementaram a utilização da Chamada de Procedimento Remoto (RPC) [41]. Para isso, adotou-se o Apache Avro [23] pois ele realiza a comunicação de forma transparente, permitindo a chamada de procedimentos que estão localizadas em outras máquinas, sem que o usuário perceba [82]. E para auxiliar na organização e na coordenação do BioNimbuZ, utilizaram um serviço voltado à sistemas distribuídos chamado ZooKeeper Apache [25] e o protocolo SFTP [36], para transferências dos arquivos. Além disso, Moura *et al.* [56] implementaram uma política de armazenamento que considera a latência e o local em que o serviço pode ser executado.

Após essa evolução, Azevedo e Freitas Junior [3] melhoraram a política de armazenamento, à qual passou a considerar a quebra de arquivos e o cáculo da largura de banda entre o cliente/servidor em sua decisão de compactação de arquivos, e de transferência de arquivos.

Entretanto, a plataforma ainda não possuía uma interface gráfica amigável. Com o intuito de melhorar a interação cliente/servidor, Ramos [68] implementou a interface gráfica e um controlador de *jobs*, responsável por fazer a ligação entre a Camada de Interface com o usuário e o núcleo do BioNimbuZ.

Ainda na linha histórica, com o objetivo de melhor distribuir as tarefas da plataforma, Barreiros Júnior [6] implementou um novo escalonador chamado de C99, que se baseia no algoritmo de busca combinacional *beam search*, levando em consideração o custo por hora dos recursos a serem alocados.

Recentemente, com a necessidade de melhorar o armazenamento dos arquivos e aproveitar o aumento na variedade e na qualidade dos serviços oferecidos pelas provedores de nuvem, Santos [76] modificou o serviço de armazenamento, para que fosse possível utilizar o serviço de armazenamento em nuvem ofertados pelos provedores de nuvem.

Com a implementação da interface gráfica, o novo escalonador e a melhoria no serviço de armazenamento, a plataforma sofreu algumas mudanças, resultando na arquitetura apresentada na próxima seção.

3.2 Arquitetura do BioNimbuZ

O BioNimbuZ faz uso de uma arquitetura hierárquica distribuída, conforme apresentada na Figura 3.1. Ela representa a interação entre as quatro camadas principais: Aplicação, Integração, Núcleo e Infraestrutura.

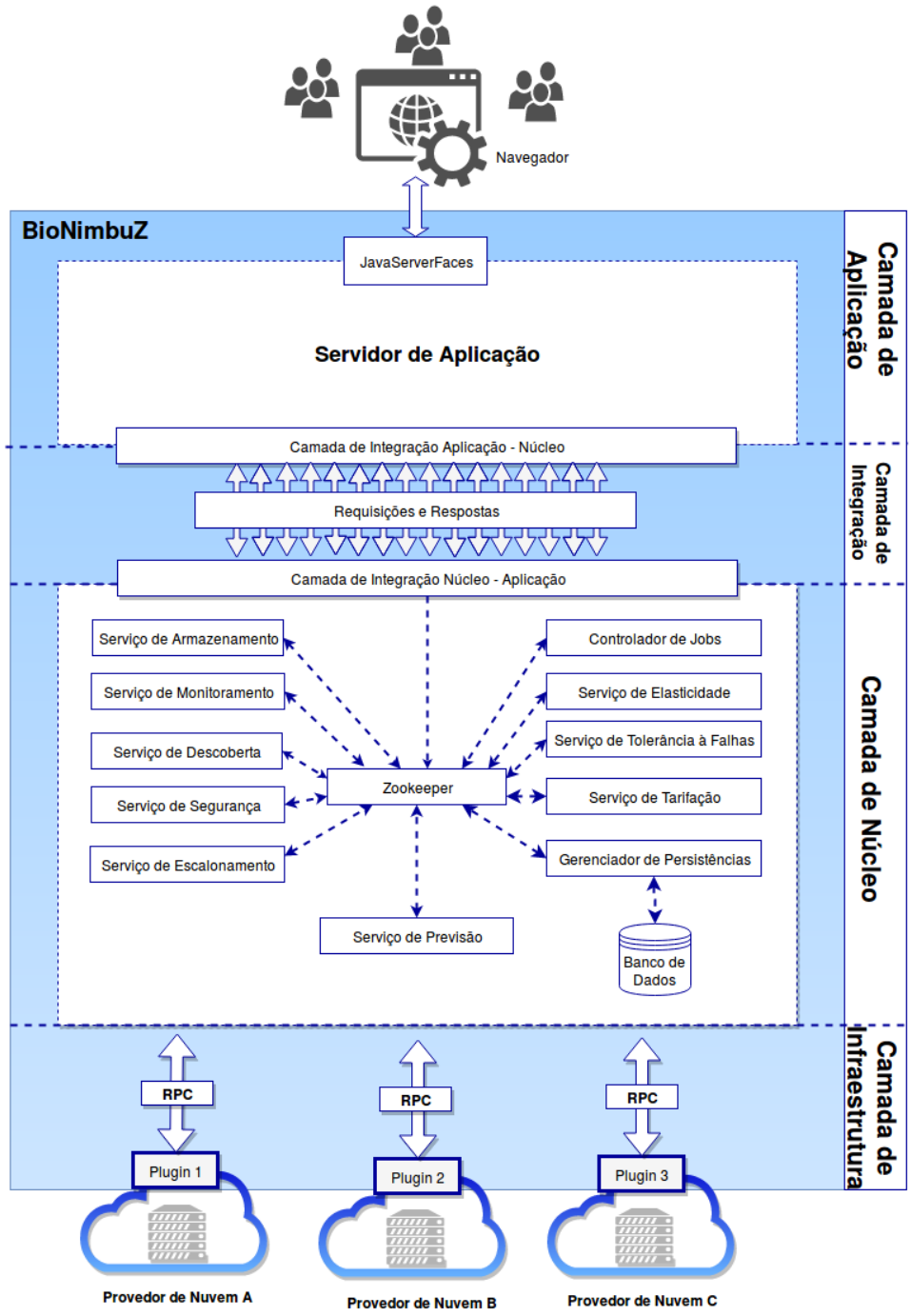

Figura 3.1: Arquitetura da Plataforma BioNimbuZ.

A primeira camada - Camada de Aplicação, permite a integração entre a aplicação

do usuário e os serviços do núcleo da plataforma. A interface com o usuário ocorre por meio de uma interface gráfica, e é nela que devem ser inseridos os *workflows* que serão executados nas diversas nuvens.

A aplicação então se comunica com o núcleo, por meio da segunda camada, a Camada de Integração, que é responsável pela transição dos dados entre a aplicação e o núcleo. A terceira camada, Camada de Núcleo, é a responsável por realizar toda a gerência da plataforma e seus serviços, e por se comunicar com os *plugins* de cada provedor, que estão na quarta camada, a Camada de Infraestrutura. Os *plugins* mapeiam as requisições e as enviam para cada provedor. A seguir serão descritos o funcionamento e as características de cada uma destas camadas em detalhes.

3.2.1 Camada de Aplicação

A Camada de Aplicação é a responsável por fazer toda a comunicação com o usuário. Por meio dessa interface, os usuários devem inserir as ações que desejam executar na federação. A interação com o usuário é realizada por meio de páginas web, ou seja, interface gráfica (GUI).

Atualmente, esta camada é composta por uma aplicação *web* que implementa um sistema gerenciador de *workflows* científicos. Essa interação inclui tarefas como *uploads* de arquivos e envio de *workflows* a serem executados. Além disso, também é responsabilidade da Camada de Aplicação exibir informações de *feedback* do sistema para o usuário, como listagem de arquivos armazenados e informações sobre uma determinada execução, tais como tempo, preço e horário de término, conforme a interface apresentada nas Figuras 3.2, 3.3, 3.4, 3.5 e 3.6.

Na Figura 3.2, pode-se observar a tela inicial do sistema, que é a primeira tela após o *login* do usuário. Nessa tela, e em todas as outras, é mostrado um *menu* com as seguintes funcionalidades disponíveis ao usuário:

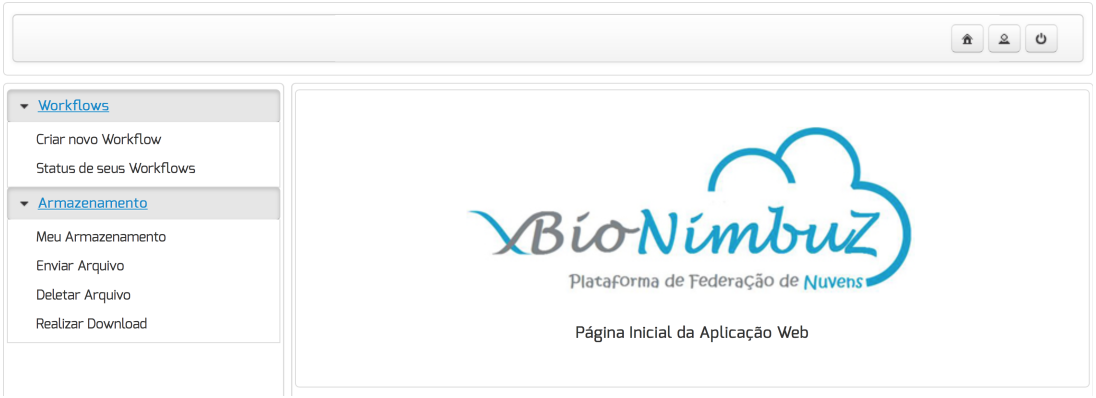

Figura 3.2: Tela Inicial da Aplicação *Web* do BioNimbuZ [68].

- ***Workflows*:** funcionalidades referentes ao gerenciamento dos *workflows* do usuário, tais como:
  - **Criar novo *Workflow*:** a partir desta opção, o usuário pode iniciar ou importar um *workflow*, no formato gerado pelo sistema;
  - ***Status* de seus *Workflows*:** possibilita ao usuário visualizar o estado e o histórico de execução de seus *workflows*.
- **Armazenamento:** funcionalidades referentes ao gerenciamento do armazenamento do usuário, sendo elas:
  - **Meu Armazenamento:** nessa opção, o usuário pode verificar a utilização de sua cota de armazenamento (definida como 1 *Gigabyte*);
  - **Enviar Arquivo:** possibilita ao usuário realizar o *upload* de arquivos à plataforma;
  - **Deletar Arquivo:** opção referente à deleção de arquivos do usuário;
  - **Realizar *Download*:** é usada caso o usuário deseje fazer o *download* dos arquivos enviados à plataforma, ou dos arquivos gerados como saída de seus *workflows*.

Ao clicar na opção, "Criar novo *Workflow*" do *menu* apresentado na Figura 3.2, o usuário é direcionado para uma outra tela no qual ele iniciará a criação do novo *workflow*, preenchendo a descrição, e selecionando os serviços que serão executados no *workflow*. Assim, após a escolha dos serviços, o usuário é redirecionado para a tela seguinte (mostrada na Figura 3.3), passando para a fase de *design* do *workflow*. Nela, são apresentados os elementos escolhidos pelo usuário dispostos de maneira gráfica, conforme apresentada na Figura 3.3.

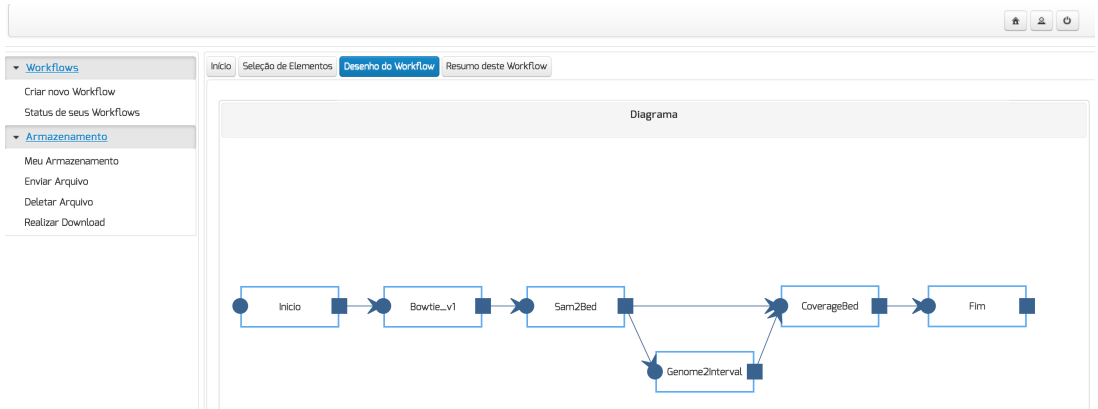

Figura 3.3: Tela de Montagem de Fluxo do *Workflow* da Aplicação [68].

Com o objetivo de compor um *workflow*, o usuário precisa fornecer alguns dados, e um desses dados são os arquivos de entrada, que são o objetivo de análise para a execução do *workflow*. Para isso então, deve-se enviar seus arquivos de entrada para a plataforma do BioNimbuZ, a partir da opção "Enviar Arquivo" do *menu* apresentado na Figura 3.2, que redireciona para a tela mostrada na Figura 3.4, possibilitando o envio desses arquivos.

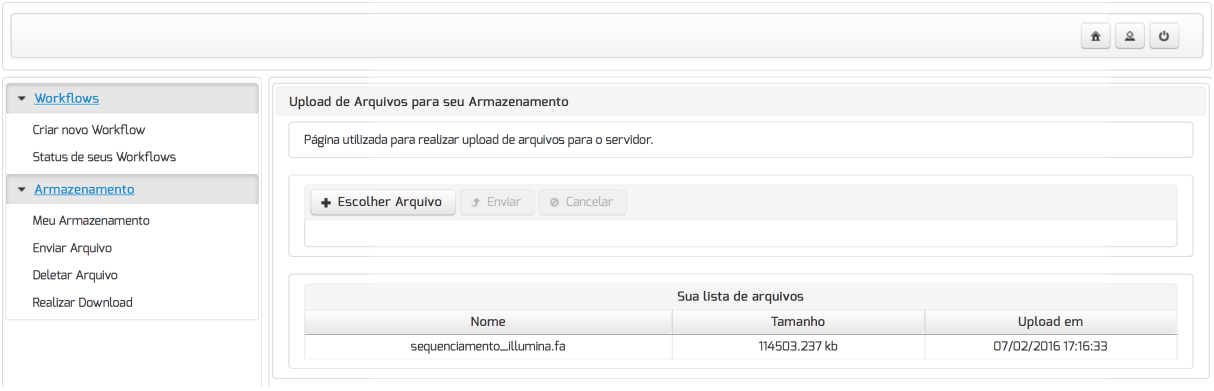

Figura 3.4: Tela de *Upload* de Arquivos [68].

A partir do momento em que o usuário escolheu submeter o *workflow* à plataforma BioNimbuZ, ele pode acompanhar o estado de sua execução na tela de *status*, selecionando a opção "*Status* de seus *Workflows*" do *menu* apresentado na Figura 3.2, permitindo assim que o usuário visualize o *status* de cada *workflow* submetido por ele, conforme é apresentado na Figura 3.5.

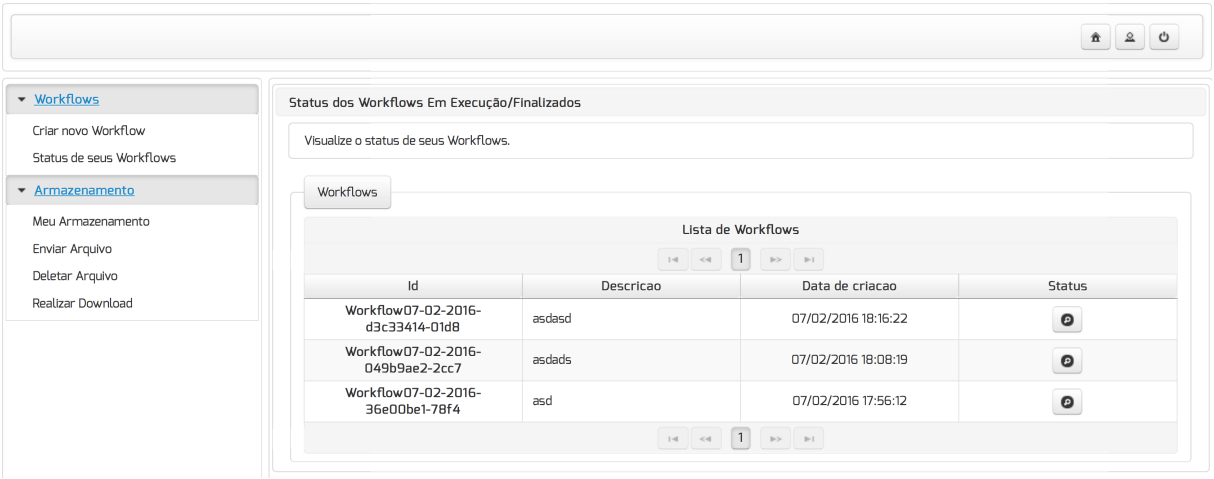

Figura 3.5: Tela de Listagem das Execuções dos *Workflows* do Usuário [68].

E para o usuário rastrear a execução de determinado *workflow*, e realizar o *download* dos arquivos de saída gerados pela execução do *workflow*, a tela apresentada na Figura 3.6 possui *logs* e também os arquivos de saída disponíveis para *download*, possibilitando esse acompanhamento mais detalhado da execução. Ao clicar em "*Download*", o navegador do usuário inicia a transferência do arquivo do BioNimbuZ para a máquina local do usuário.

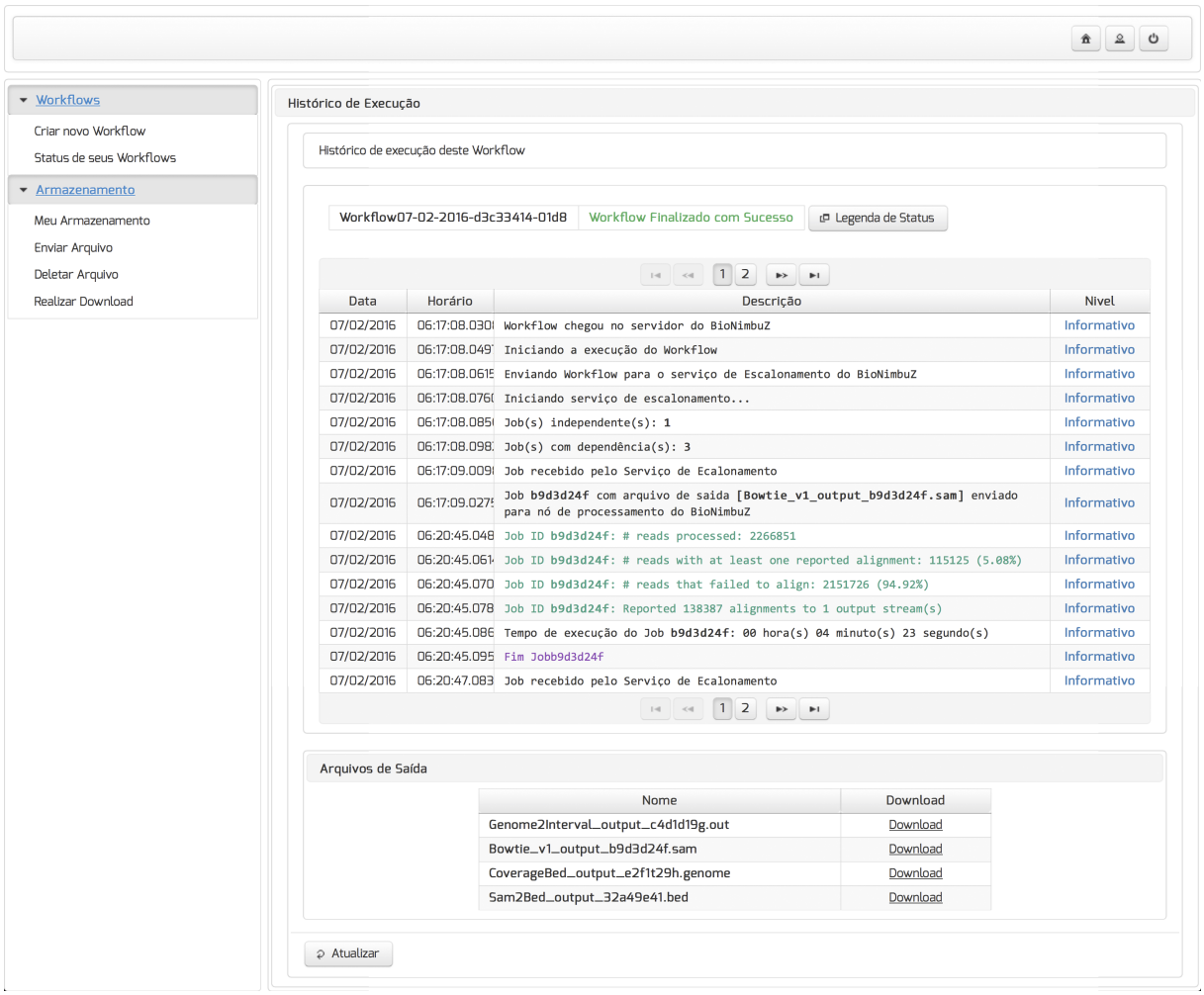

Figura 3.6: Tela de Monitoramento da Execução do *Workflow* [68].

3.2.2 Camada de Integração

A Camada de Integração é a responsável por integrar as Camadas de Aplicação e de Núcleo. Ela realiza esta tarefa por meio da troca de mensagens entre estas camadas, via serviços *web*. Assim, a Camada de Integração permite disparar requisições da Camada de Aplicação para o Núcleo, e receber respostas do Núcleo para a Camada de Aplicação.

Para tratar as requisições recebidas da Camada de Aplicação e enviar respostas para ela, utiliza-se um *framework* REST, chamado Resteasy [69]. A principal vantagem ao se desenvolver interfaces baseadas em REST é que sistemas com a capacidade de enviar requisições HTTP pela Internet, podem ser integrados a outros sistema com essas mesmas características. Dessa forma, a interface desenvolvida na Camada de Núcleo do BioNimbuZ é independente da aplicação que irá acessá-lo [68].

3.2.3 Camada de Núcleo

A Camada de Núcleo, também conhecida apenas por Núcleo, é a responsável por toda a gerência da federação, tendo como atividades o descobrimento de provedores e recursos, o escalonamento de tarefas, o gerenciamento de tarefas, o armazenamento de arquivos e o controle de acesso dos usuários, entre outras. Para cada uma destas atividades existe um serviço responsável. A seguir serão descritos os serviços que contemplam o Núcleo da plataforma BioNimbuZ:

- Controlador de *Jobs*: é quem recebe a requisição que vem da aplicação, verificando as credenciais dos usuários com o serviço de segurança, para permitir ou não a execução das tarefas. Além disso, o Controlador de *Jobs* é o responsável por gerenciar os vários pedidos e garantir que os resultados sejam entregues de forma correta para cada uma das requisições, e mantê-los para que possam ser consultados posteriormente [68];
- Serviço de Elasticidade: é o serviço responsável por dinamicamente aumentar ou diminuir o número de instâncias de máquinas virtuais, ou então reconfigurar os parâmetros de utilização de CPU, de memória, de largura de banda, entre outros recursos que são disponibilizadas pelos provedores de nuvem, a fim de obter uma melhor utilização da infraestrutura disponível. A elasticidade pode ser vertical, quando há um redimensionamento dos atributos de CPU, de armazenamento, de rede ou de memória, como também pode ser horizontal, quando o número de instâncias de máquinas virtuais é modificado para mais ou para menos

  Para tomar as ações de elasticidade do BioNimbuZ, tem-se um controlador de elasticidade, como é apresentado na Figura 3.7.

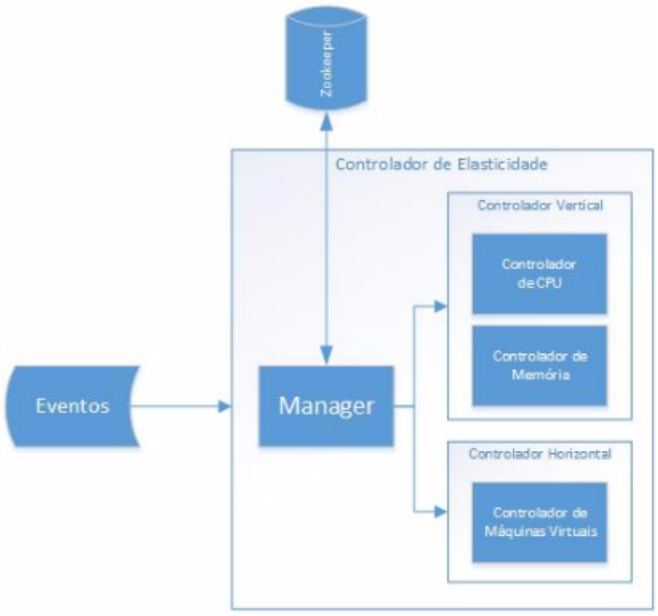

Figura 3.7: Arquitetura do Serviço de Elasticidade [87].

Esse controlador de elasticidade, desenvolvido por Vergara [87], possui três módulos principais, os quais são o *Manager*, o Controlador de Elasticidade Horizontal e o Controlador de Elasticidade Vertical. O *Manager* é o responsável por receber os eventos de disparo da elasticidade, por decidir entre elasticidade horizontal ou vertical, e por atualizar o ZooKeeper [25] com as informações das máquinas virtuais. O segundo módulo é o Controlador de Elasticidade Horizontal, o qual é responsável por escolher qual o tipo de máquina virtual deve ser criada e instância-la. Além disso, ele também é responsável por remover máquinas virtuais que estão inutilizadas. Por último, tem-se o Controlador de Elasticidade Vertical, responsável por aumentar/diminuir o número de CPU e memória das máquinas virtuais;

- Serviço de Monitoramento: é o serviço responsável por realizar um acompanhamento dos recursos da federação, junto ao ZooKeeper [25], com a utilização de *watchers* é capaz de tratar os alertas enviados pelos mesmos. Uma outra responsabilidade deste serviço é permitir a recuperação dos dados principais utilizados pelos módulos de cada servidor BioNimbuZ, armazenados na estrutura do ZooKeeper, para possíveis reconstruções ou garantias de execuções de serviços solicitados;
- Serviço de Descobrimento: é o serviço que identifica e mantém informações a respeito dos provedores de nuvem que estão na federação, tais como capacidade de

armazenamento, processamento e latência de rede, e também mantém detalhes sobre parâmetros de execução e arquivos de entrada e de saída. Para obter estas informações a respeito dos provedores, o ZooKeeper é consultado, pois todos os provedores presentes na federação possuem *znodes* [25] que armazenam seus dados. Assim, sempre que uma modificação é realizada, como a saída ou a entrada de algum provedor, os *watchers* [25] alertam os outros serviços mantendo, assim, todos os participantes da federação atualizados;

- Serviço de Escalonamento: é o serviço que recebe o pedido para a execução de alguma tarefa - aqui chamado de *job* - e as distribui dinamicamente entre os provedores disponíveis, divididas em instâncias menores - *tasks*. O Serviço de Escalonamento também é responsável por acompanhar toda a execução do *job*, e manter um registro das execuções já escalonadas.

  Para realizar a distribuição de tarefas na federação algumas métricas são levadas em consideração, como latência, balanceamento de carga, tempo de espera e capacidade de processamento. O Serviço de Escalonamento do BioNimbuZ possui algumas políticas de escalonamento, as quais são a ACOSched [61] e a C99 [6]. Ambas implementam técnicas de escalonamento distintas, sendo que o escalonador usado atualmente na plataforma BioNimbuZ é o a C99 [6]. O algoritmo C99 tem por objetivo ser *any-time* e incremental com uma rápida convergência para boas soluções. Por incremental e *any-time* entende-se que o algoritmo convergirá para um melhor conjunto de soluções com o decorrer de sua execução, e que o algoritmo poderá ser parado em qualquer momento da sua execução, retornando ainda um conjunto de boas soluções, mais detalhes sobre o C99 podem ser vistos no trabalho de Barreiros Júnior [6];

- Serviço de Armazenamento: responsável pela estratégia de armazenamento dos arquivos que são utilizados ou mantidos pelas aplicações. O armazenamento deve ocorrer de forma eficiente para que as aplicações possam utilizar os arquivos com o menor custo possível. Este custo é calculado utilizando-se algumas métricas, tais como latência de rede, distância entre os provedores e capacidade de armazenamento. O Serviço de Armazenamento foi, inicialmente, implementado com o trabalho de Bacelar e Moura [57], em paralelo, Gallon [28] desenvolveu outra política que considerava alguns critérios a mais para a escolha da melhor nuvem para o armazenamento dos arquivos. Esses critérios são, além da latência, o custo de armazenamento, os núcleos de processadores e a carga de trabalho. Em seguida, a política de armazenamento foi aprimorada por Azevedo e Freitas Junior [3], adicionando a largura de banda e a verificação da necessidade de se compactar o arquivo

que será realizado o *upload* na métrica de cálculos, e nas métricas de decisão de envio do arquivo.

Com a necessidade de melhorar o modo de armazenar e de dispor os arquivos, Santos [76] adicionou ao serviço de armazenamento um módulo que faz uso dos serviços de armazenamento em nuvem, possibilitando assim que o serviço de armazenamento tenha dois modos de armazenamento. O primeiro modo é o armazenamento nas próprias máquinas virtuais que estão executando o *workflow*. O segundo modo utiliza-se da tecnologia dos serviços de armazenamento por objeto [16], oferecidos pelos provedores externos utilizados na federação mais especificamente, o Amazon S3 [46] e o Google *Cloud Storage* [29]. Nesse segundo modo o armazenamento dos arquivos é realizado em *buckets*, que são volumes nos quais os arquivos são armazenados. Desta forma, requisições internas de arquivo são atendidas através da transferência do arquivo solicitado entre um dos *buckets* contendo o arquivo e a máquina virtual que vai realizar a requisição.

Outra estratégia adotada no armazenamento dos arquivos é a replicação em mais de um provedor, garantindo que os arquivos estejam disponíveis quando as aplicações forem utilizá-los;

- Serviço de Tolerância a Falhas: tem como objetivo garantir que todos os principais serviços estejam sempre disponíveis. Assim, em caso de falhas, garantir que os serviços sejam recuperados. Essa recuperação exige que todos os objetos que forem afetados pela falha voltem ao estado anterior, sem falhas. O serviço de tolerância a falhas possui atuação de forma distribuída na plataforma BioNimbuZ, e está presente em outros serviços.

  No Serviço de Armazenamento, por exemplo, quando um alerta de indisponibilidade de um recurso é lançado por um *watcher*, é iniciada uma rotina de recuperação para os arquivos que este recurso continha. Como os arquivos são armazenados de forma duplicada na federação, a recuperação ocorre de forma a identificar quais arquivos foram perdidos e realizar uma nova duplicação em outro servidor do BioNimbuZ.

  No Serviço de Escalonamento, a recuperação está presente quando é recebido um alerta de indisponibilidade de um determinado recurso, e todas as tarefas que foram escalonadas e ainda não haviam sido executadas, ou estavam em execução, são novamente escalonadas, agora para outras máquinas na federação.

  Essas recuperações são possíveis devido aos *watchers*, que disparam alertas aos responsáveis, avisando sobre os problemas em algum recurso da federação.

- Serviço de Tarifação: é o serviço responsável por calcular o quanto os usuários devem pagar pela utilização dos serviços oferecidos na plataforma BioNimbuZ. Para que isto seja possível, este serviço se mantém em constante contato com o Serviço de Monitoramento, obtendo informações tais como tempo de execução e quantidade de máquinas virtuais alocadas para as tarefas que foram, e que estão sendo executadas pelo usuário. Em um ambiente de nuvem federada, a complexidade de tal serviço é maior do que em um ambiente de nuvens computacionais, uma vez que a infraestrutura de uma plataforma de nuvens federadas é garantida pela recepção de recursos de diversas nuvens autônomas.

  Assim, no BioNimbuZ é função do Serviço de Tarifação garantir o cumprimento das métricas de tarifação das nuvens independentes, definidas nos contratos assinados entre as nuvens provedoras e a plataforma, e realizar a cobrança baseada na demanda do cliente, de maneira que não haja percepção por parte do cliente de que há recursos de diversas nuvens. Diante do exposto, foi definida uma arquitetura para o Serviço de Tarifação do BioNimbuZ, arquitetura essa que pode ser observada na Figura 3.8.

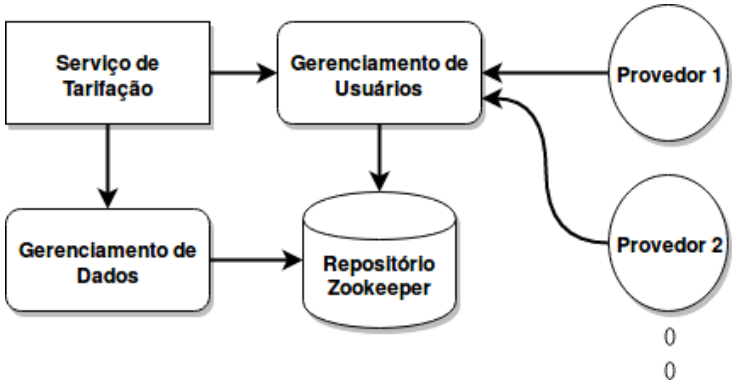

Figura 3.8: Arquitetura do Serviço de Tarifação do BioNimbuZ.

O componente Gerenciamento de Dados é o responsável pela obtenção das métricas de tarifação das nuvens que compõem a Camada de Infraestrutura do BioNimbuZ. O componente Gerenciamento de Usuário é o componente responsável por cumprir as métricas de tarifação definidas nos contratos firmados com as nuvens fornecedoras de recurso, e realizar a cobrança dos clientes baseada na demanda de suas aplicações. Por fim, o ZooKeeper é utilizado como mecanismo de persistência, para que os dados adquiridos pelos componentes de Gerenciamento de Dados e de Gerenciamento de Usuário, sejam mantidos no ambiente. Dessa maneira, todas as informações do Serviço de Tarifação ficam disponíveis para que os outros serviços possam usá-las.

- Serviço de Segurança: segurança em nuvens federadas é uma área de estudo em constante evolução, e o serviço de segurança deve trabalhar em diversos pontos

para fornecer um serviço efetivamente seguro. O primeiro passo é a autenticação de usuários, ou seja, é preciso saber se o usuário que está tentando acessar algum recurso na federação é quem ele realmente diz ser. Após a autenticação, vem a autorização, que consiste em verificar se o usuário pode realizar as ações que deseja. Muitos outros aspectos podem ser abordados por este serviço, como a criptografia de mensagens, para garantir a confidencialidade na troca de informações entre provedores; e também a verificação de integridade de arquivos, de modo que seja possível garantir que um arquivo não seja alterado por fatores externos à federação [68] [77].

- Serviço de Predição: qualquer *workflow* submetido à federação possui custos computacionais e financeiros os quais devem ser transparentes para o usuário. Desta forma, o BioNimBuZ possui o Serviço de Predição de Custos e Recursos Computacionais - sPCR [73], com o objetivo de auxiliar o usuário na escolha de um ambiente computacional que execute seus *workflows* científicos de forma transparente, com estimativas de tempo, custo financeiro e recursos a serem utilizados. O usuário preenche um *template* com as informações do *workflow* a ser executado, para que o serviço estime o custo, tempo e os melhores recursos computacionais que se adequem as variáveis informadas. A Figura 3.9 define a estrutura e o funcionamento do Serviço de Predição, que é realizado em quatro fases.

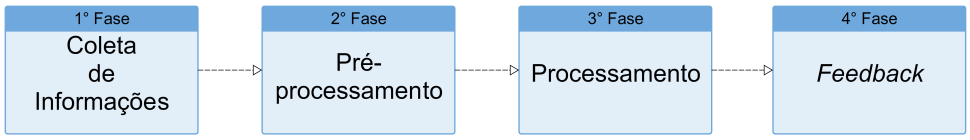

Figura 3.9: Fases de Execução do sPCR [73].

A primeira fase consiste na coleta das informações do *workflow* a ser executado, estas informações são fornecidas por meio da interface com o usuário, no qual o mesmo irá inserir a estrutura do *workflow* a ser executado, juntamente com as restrições de custo financeiro, tempo máximo de duração da execução, definição entre uma execução de baixo custo ou de alto desempenho. Além disso, defini-se parâmetros de recursos computacionais, tais como capacidade de memória RAM, disco e poder de processamento. Estas informações irão especificar a lista de recursos compatíveis com as restrições impostas.

A segunda fase é composta pela base de dados histórica e pela comunicação com o Serviço de Tarifação do ambiente de nuvem. A base de dados histórica irá auxiliar na acurácia da escolha dos recursos, na estimativa do tempo de execução e do custo financeiro. A base é composta por informações de execuções passadas, na qual são

armazenados detalhes do *workflow*, tais como softwares utilizados, arquivos envolvidos, dimensão dos dados, tempo de execução de cada fase, consumo computacional de cada fase e etc. Logo, é possível utilizar algoritmos que avaliem a base histórica a procura de execuções similares, para aprimorar a escolha do recurso e nas estimativas de custo e tempo. E a comunicação com o Serviço de Tarifação é realizada de forma em que o serviço informa a lista de recursos disponíveis, indicando a capacidade computacional de cada máquina virtual, custos financeiros e detalhes do provedor. Com isso, esta fase tem o objetivo de coletar informações descritas pelo usuário e do ambiente computacional, com o intuito de reportar as informações necessárias para a terceira fase do processo de predição.

A terceira fase é composta por um algoritmo baseado na abordagem GRASP [18], que consome as informações da fase anterior, com o objetivo de minimizar custos e tempo de execução, levando em consideração as definições do usuário. O objetivo desta fase é que, por meio da metaheurística, seja possível disponibilizar boas soluções, em tempo viável.

Após a construção da solução, são enviados para a interface do usuário os *feedbacks* das soluções encontradas e as estimativas de tempo e custo da execução, completando assim a quarta e última fase da predição, auxiliando assim, o usuário em uma escolha de recursos para executar seu *workflow*, que as vezes não é trivial. Além destas informações, a fase de *Feedback* alimenta a base de dados histórica, a fim de registrar os dados da predição, e assim melhorar as métricas da predição.

E para realizar a comunicação de forma transparente para o usuário, permitindo assim a execução de chamadas de procedimento, o sistema de RPC e de serialização, o Apache Avro [23] é utilizado. E no auxílio da organização e na coordenação do BioNimbuZ, utiliza-se um serviço voltado à sistemas distribuídos chamado Apache ZooKeeper [25]. Esses serviços são apresentados a seguir.

Apache Avro

O Apache Avro [23] é um sistema de RPC e de serialização de dados desenvolvido pela Fundação Apache [27]. Algumas vantagens deste sistema são o fato de ser livre, a utilização de mais de um protocolo de transporte de dados em rede, e o suporte a mais de um formato de serialização de dados. Quanto ao formato dos dados, existe o suporte aos dados binários e aos dados no formato JSON [15]. Em relação aos protocolos, pode-se utilizar tanto o protocolo HTTP [19] ou o protocolo próprio do Avro, que é o protocolo RPC [26].

O Avro foi criado para ser utilizado com um grande volume de dados, e possui algumas características [26] definidas pela própria Fundação Apache, tais como uma rica estrutura de dados com tipos primitivos, um formato de dados compacto, rápido e binário e a integração de forma simples com diversas linguagens de programação.

O Avro foi escolhido como *middleware* de Chamada Remota de Procedimento para o BioNimbuZ, por ser livre, flexível e possuir integração com várias linguagens. Ele funciona de modo em que as requisições de execuções de procedimentos, sejam serializadas para a execução remota e ao receber a requisição, a máquina responsável pela execução recebe essa execução já deserializada.

Apache ZooKeeper

O Apache ZooKeeper [25], ou apenas ZooKeeper, é um serviço de coordenação de sistemas distribuídos, criado pela Fundação Apache, para ser de fácil manuseio. Ele utiliza um modelo de dados que simula uma estrutura de diretórios, e tem como finalidade facilitar a criação e a gestão de sistemas distribuídos, que podem ser de alta complexidade e de difícil coordenação e manutenção.

No ZooKeeper são utilizados espaços de nomes chamados de *znodes*, que são organizados de forma hierárquica, assim como ocorre nos sistemas de arquivos. Cada *znode* tem a capacidade de armazenar no máximo 1 Megabyte (MB) de informação, e são identificados pelo seu caminho na estrutura. Neles podem ser armazenadas informações que facilitem o controle do sistema distribuído, tais como metadados, caminhos, dados de configuração e endereços [39].

No ZooKeeper existem dois tipos de *znodes*, os persistentes e os efêmeros. O primeiro é aquele que continua a existir mesmo depois da queda de um provedor, sendo útil para armazenar informações a respeito dos *jobs* que foram executados e outros tipos de dados que não se deseja perder. Os efêmeros, por outro lado, podem ser criados para cada novo participante da federação, pois quando este novo participante ficar indisponível, o *znode* efêmero referente a ele será eliminado e todos os outros componentes do sistema distribuído saberão que ele não se encontra disponível. Na Figura 3.10 pode ser visto um exemplo da estrutura hierárquica dos *znodes* que foi implementada no BioNimbuZ, no qual os metadados da aplicação são armazenados de maneira em que seja eficiente a forma de recuperação de informação da federação.

Assim, como pode ser observado na Figura 3.10, o BioNimbuZ cria uma estrutura lógica da aplicação, de forma a receber as informações necessárias para a execução das tarefas de forma dinâmica e transparente, em que existem *znodes* persistentes, para armazenar as informações importantes, que não podem ser apagadas caso alguma máquina fique indisponível; e um nó efêmero, que ativa uma *trigger*, chamada de *watcher*, para

que os serviços recuperem as tarefas e os arquivos que estavam na máquina que ficou indisponível.

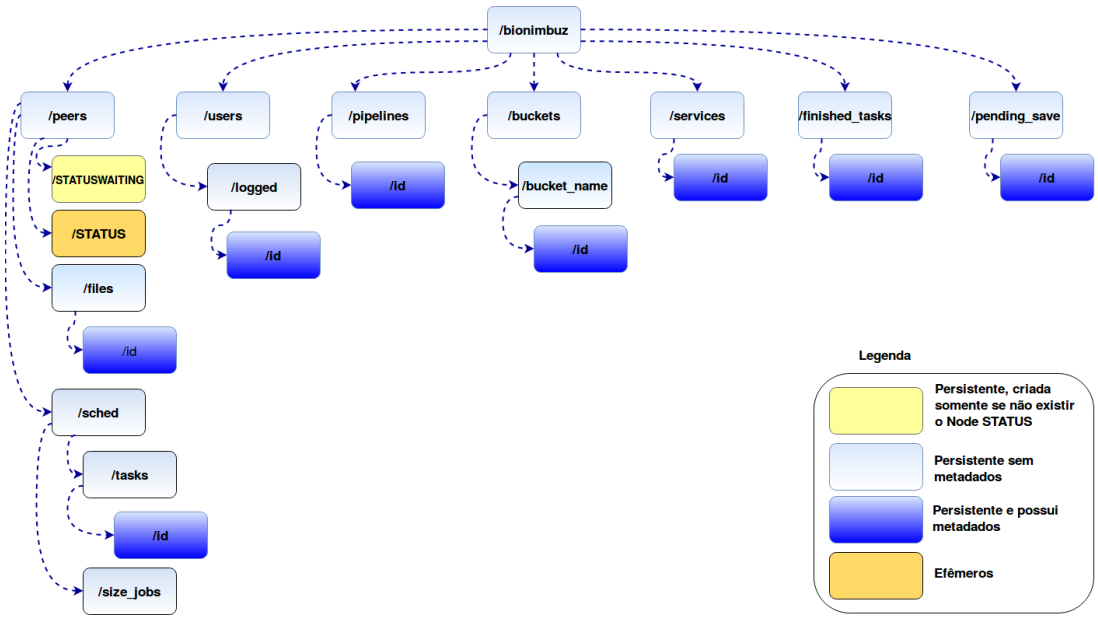

Figura 3.10: Estrutura Hierárquica dos *Znodes* no BioNimbuZ [68].

Os *watchers* funcionam como observadores de mudanças, e enviam alertas sobre as alterações ocorridas em algum dos *znodes*. Este conceito é útil para sistemas distribuídos, pois permitem o monitoramento constante do sistema, deixando que os *watchers* avisem quando um provedor estiver fora do ar, por exemplo, ou então quando um novo recurso estiver disponível. Na plataforma BioNimbuZ esses *watchers* são instanciados pelos serviços e controladores, por meio do *framework* chamado Curator [93], o qual é disponibilizado pelo Apache para facilitar o *CRUD*(*Create Read Update Delete*) de *znodes*, e o instanciamento dos *watchers* para os serviços no ZooKeeper [93].

3.2.4 Camada de Infraestrutura

A camada de infraestrutura consiste em todos os recursos que os provedores de nuvens colocam a disposição da federação, somados aos seus respectivos *plugins*. O principal objetivo desta camada é prover uma interface de comunicação entre o BioNimbuZ e os provedores de nuvens, utilizando para tal os *plugins* que mapeiam as requisições provenientes da arquitetura, para os comandos específicos de cada provedor, individualmente.

Assim sendo, faz-se necessário desenvolver um *plugin* para cada plataforma de nuvem, utilizando a infraestrutura como serviço, tais como o *Elastic Compute Cloud* (EC2) [45] e o *Google Compute Engine* [33], serviços ofertados respectativamente pela Amazon [78] e pela Google [31].

Atualmente, a plataforma BioNimbuZ tem sido amplamente usada para a execução de *workflows* científicos de Bioinformática, que será detalhado na próxima seção.

3.3 *Workflows* Científicos

O termo *workflow* originou-se na década de 70, associados aos processos de automação de escritórios, que tinha como objetivo oferecer soluções para diminuir a geração e a distribuição de documentos em papel em uma organização. Neste contexto, um *workflow* pode ser entendido como a automação total ou parcial de um processo, na qual informações ou tarefas são passadas de uma entidade para outra de acordo com um conjunto de regras [85].

Mais recentemente, o conceito de *workflow* vem sendo aplicado às ciências, na automação de experimentos computacionais que necessitam de grande poder de processamento e manipulam grande quantidade de dados, possivelmente distribuídos. Nesse contexto, o termo passou a ser chamado de *workflow* científico, que de acordo com Singh *et al.* [1] é uma série de atividades estruturadas e cálculos que surgem na resolução de problemas científicos. Assim, ele pode ser visto como um processo automatizado que combina dados e processos em um conjunto estruturado de passos para implementar soluções computacionais para um problema científico.

No contexto da Bioinformática, as análises computacionais dos dados, obtidos por meio de sequenciadores automáticos, são realizadas em diferentes fases ou passos. Para cada fase existe um conjunto de ferramentas de Bioinformática a serem utilizadas. Entretanto, cada tipo de pesquisa implica numa combinação diferente de ferramentas, de acordo com os objetivos da pesquisa, e este fluxo de passos é chamado de *workflow* de Bioinformática [68].

Assim sendo, *workflows* de Bioinformática estão relacionados ao processamento de dados biológicos e, em geral, aos processos de sequenciamento de DNA (Ácido Desoxirribonucléico) ou RNA (Ácido Ribonucleico). Estes processos se relacionam na descoberta de qual é a sequência de bases que forma cada fragmento de DNA ou RNA de um determinado organismo que está sendo investigado. A partir desses fragmentos, tanto de DNA quanto de RNA, vários processos computacionais podem ser executados de acordo com os objetivos do projeto [34].

De forma geral, projetos de Bioinformática possuem suporte computacional, nos quais são projetados *workflow* que transformam fragmentos de entrada, de forma a extrair informações como funções biológicas e localização dentro da célula. O exemplo da Figura 3.11 mostra um *workflow* de três fases, as quais são filtragem, mapeamento/montagem e análise, descritas em detalhes a seguir.

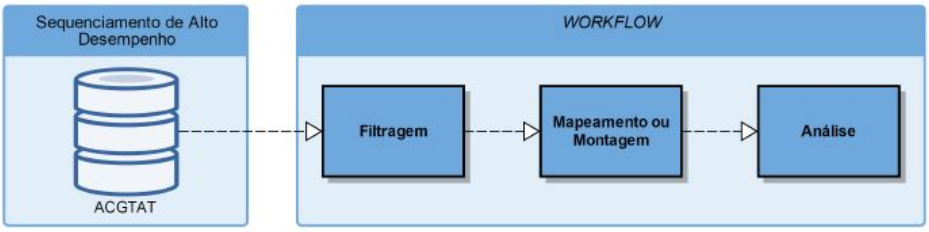

Figura 3.11: Exemplo de *Workflow* de Bioinformática [63].

Os biólogos inicialmente realizam processos laboratoriais de coleta e replicação do material biológico. Em seguida, é realizado o sequenciamento de pequenas porções de DNA ou RNA (de acordo com o projeto), que são denominadas *reads* [34]. As *reads* coletadas passam para o processo de filtragem, que elimina as *reads* de qualidade inferior ou parte delas, com um certo grau de confiabilidade mínima.

Na fase de mapeamento (ou montagem), o objetivo é localizar em um genoma de referência o alinhamento para o maior número de *reads* filtradas. O término da fase de sequenciamento é realizado na fase de análise, na qual tem-se grandes porções do DNA ou RNA sequenciados, que são analisados por diferentes processos que dependem do objetivo do experimento.

Como pode ser percebido, todo esse processo de execução de *workflows* de Bioinformática demanda expressivos recursos computacionais. Para isso, pode-se utilizar o poder computacional disponível na computação em nuvem.

3.4 Considerações Finais

Neste capítulo foi descrita a plataforma de federação em nuvens BioNimbuZ. Para isso, foram apresentadas suas evoluções, sua arquitetura divida em camadas e seus detalhes de funcionamento.

Contudo, não adianta um sistema ser escalável, flexível e tolerante a falhas, se ele não garante a qualidade de serviço a ser entregue para o usuário. Essa qualidade deve ser protegida e garantida por meio de contratos, denominados Contrato de Nível de Serviço (em inglês, *Service Level Agreement - SLA*). Nesse contexto, o próximo capítulo abordará as definições, as características e as propostas de implementação do SLA em nuvem, para uma melhor compreensão do tema desenvolvido neste trabalho, que será apresentado no Capítulo 5.

Capítulo 4

Acordo de Nível de Serviço (*Service Level Agreement - SLA*)

O objetivo deste capítulo é explicar as definições do Acordo de Nível de Serviço (*Service Level Agreement - SLA*) e apresentar o estado da arte de SLA em computação em nuvem, e em nuvens federadas. Para isso, na Seção 4.1 são apresentadas as definições de SLA, os componentes que o definem e sua arquitetura em sistemas de computação em nuvem. Em seguida, na Seção 4.2 é descrito o ciclo de vida de um SLA. Para finalizar, na Seção 4.3 são apresentados os trabalhos relacionados, e na Seção 4.4 algumas considerações finais.

4.1 Visão Geral

O nível de confiabilidade e de contentamento do cliente em computação é crucial para que sistemas tenham sucesso em seu mercado. Assim, implementar mecanismos que possam garantir a entrega e a qualidade aos serviços prestados é muito importante para um provedor de serviço ou produto, pois garante uma maior confiabilidade de seus serviços prestados ao usuário, além de obter a fidelidade do usuário. Para isso, contratos denominados de SLA, são criados a fim de que essas características sejam atendidas [38].

Os SLAs são utilizados para identificar as partes envolvidas em um negócio, além de especificar o mínimo de expectativas e limites que existe entre as partes, buscando melhorar a qualidade de serviço e a relação entre cliente e provedor [8].

Assim sendo, em todo negócio firmado entre duas partes, para que se garanta a qualidade e a entrega do serviço ou produto a ser adquirido, sempre existiu contratos. Estes contratos são firmados entre cliente e provedor de serviço ou produto, para que regras sejam estabelecidas entre as partes, garantindo que o serviço ou o produto fornecido seja entregue de acordo com os termos deste contrato.

Nesse contexto, o termo SLA surgiu nos anos 80, sendo abordado em vários âmbitos, desde serviços de Internet à gerência de centro de dados, entre outros. Segundo Verma [88], um SLA é definido por um estado explícito de expectativas e obrigações que existe em uma relação de negócios entre duas organizações, a provedora de serviço e o cliente.

Nesse cenário, em nuvem computacional, o SLA está presente em várias áreas de serviços, como é mostrado por Wu e Buyya [90]:

- *Web Service*: acordo usado para garantir a entrega dos serviços *web*, o qual define o entendimento e as expectativas do provedor de serviço e do consumidor do serviço [37];
- Serviços de Rede: contrato entre um provedor de rede e um cliente que especifica, usualmente em termos mensuráveis, quais serviços o provedor de rede irá atender e quais as penalidades serão adotadas se o provedor de serviço não atender os objetivos estabelecidos [90];
- Serviços de Internet: é a construção formal para a entrega de serviço. Todas as partes, consumidor e provedor, envolvidos são usuários do SLA. Consumidores de serviço utilizam SLA como uma descrição do contrato, de forma legal do que o provedor promete fornecer. O provedor de serviço usa o SLA para ter um contrato definitivo do que deve ser entregue [81];
- Serviços de Gerenciamento de Centro de Dados: é um acordo formal para prometer o que é possível prover, e prover o que foi prometido [90].

Sistemas de computação em nuvem englobam essas perspectivas de SLA, pois fazem uso dessas áreas para ofertar em seus serviços e, assim, tornarem-se competitivas nesse mercado.

Com o aumento de investimentos no setor da computação em nuvem, o aumento dos dados nesse setor e a crescente disputa por consumidores que utilizam serviços em nuvens, ofertas de serviços surgem cada vez mais acirradas para conquistar o consumidor desse mercado. Assim, para as prestadoras de serviços em nuvem continuarem competitivas no mercado é adequado que as mesmas formem uma federação de nuvens, pois é um meio de ofertar serviços e recursos, que as vezes possam ser menos custosos. E para garantir que esses provedores de serviços em nuvens cumpram o que estão ofertando, existem contratos que são assinados entre as partes envolvidas, no qual é iniciado o ciclo de vida do contrato, chamado de Acordo de Nível de Serviço. Nesses contratos há componentes que servem como alicerce da construção do SLA [17].

Os componentes do SLA são a base do contrato, no qual estão descritas todas as definições do acordo como os objetivos do SLA, as restrições, a validade do contrato, o

escopo do contrato, as partes envolvidas, os objetivos de nível de serviço, as penalidades em caso de quebra do contrato, os serviços opcionais caso necessite, e a administração que são os processos usados para garantir que os objetivos de nível de serviço sejam alcançados. Para garantir esses objetivos, o monitoramento de atividades é realizado para que o acordo seja satisfeito, garantindo a Qualidade de Serviço.

Assim, as definições desses componentes são umas das mais importantes etapas do ciclo de vida do SLA, que engloba as etapas da criação, da execução e da finalização do contrato. Todavia, para a criação do ciclo de vida de um SLA é necessário definir uma arquitetura que envolva o usuário ou o consumidor do serviço. Esses submetem as requisições por meio de interfaces ou *middlewares*, em que as interfaces se comunicam com um serviço examinador de requisições. O Serviço Examinador de Requisições é responsável pelo controle de aceitação de requisições, passando a requisição feita pelo usuário para o gerenciador de SLA ou (controlador de SLA), que irá administrar as alocações dos recursos em provedores de serviços de acordo com as requisições realizadas [90], conforme apresentado na Figura 4.1.

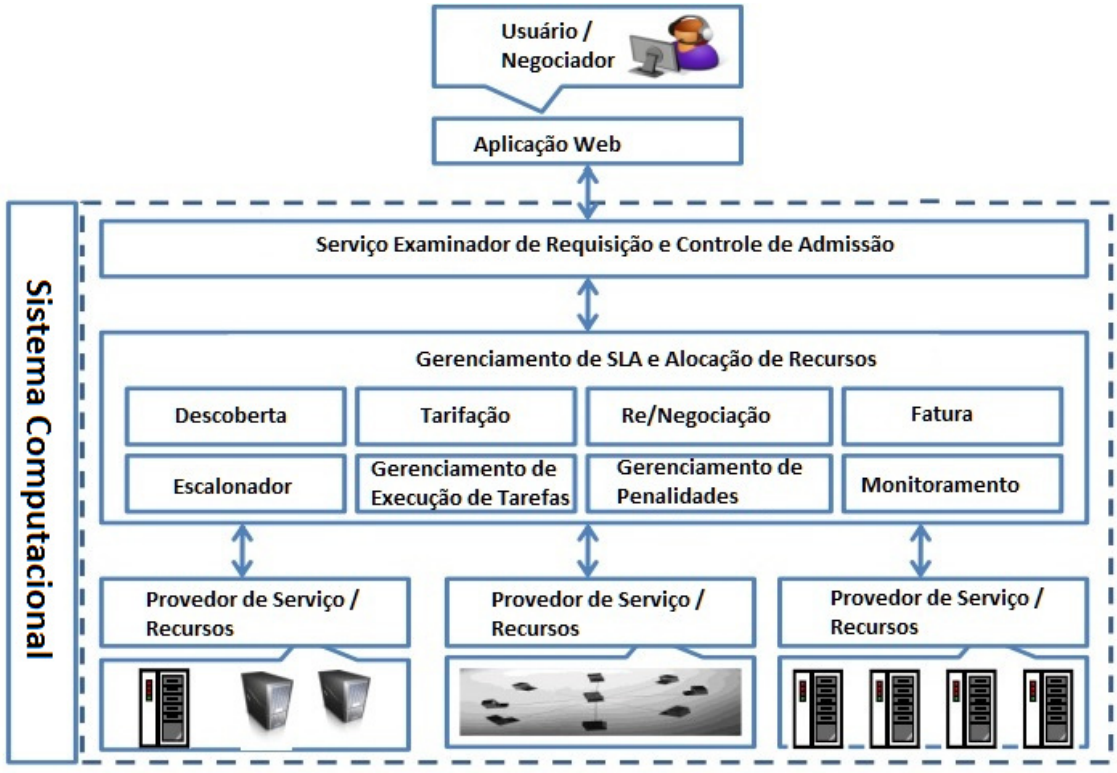

Figura 4.1: Arquitetura Geral para Sistemas Computacionais, adaptado de Wu e Buyya [90].

Como observado na Figura 4.1, a arquitetura possui alguns componentes que são

gerenciados pelo controlador de SLA, são eles: Descoberta, Tarifação, Negociação/Renegociação, Fatura, Escalonador, Monitoramento, Gerenciamento de Penalidades e Gerenciamento de Execução de Tarefas.

O componente de Descoberta é responsável por descobrir provedores de serviços que possam atender as requisições do usuário. Além de definir os termos acordados entre as partes.

O componente de Tarifação decide como será tarifado os serviços requisitados, baseando-se nos preços dos provedores de serviços para suprir e demandar recursos computacionais, facilitando, assim, a alocação dos recursos.

Uma vez que o processo de Negociação seja completado, o componente de Gerenciamento de Execução de Tarefas é acionado, disparando assim uma requisição para o componente Escalonador, esse por sua vez utiliza algoritmos e políticas para mapear os recursos às requisições.

O componente de Monitoramento consiste em um mecânismo de monitoramento de recursos, que rastreia a disponibilidade dos recursos providos e suas configurações, além de monitorar o progresso da execução das tarefas.

O mecânismo de Gerenciamento de Penalidades, gerencia os termos de contratos violados durante a execução das tarefas. Devido a violação do SLA, algumas renegociações são necessárias para manter o acordo firmado. Com isso, o componente de Fatura cálcula os custos dos recursos alocados, e gera uma conta para o usuário.

Esses componentes formam um processo que possui três fases, que são início, meio e fim. O início é formado por descoberta, definição e estabelecimento de acordo. A segunda etapa, o meio, é composto pela execução ou monitoramento do contrato. E a última etapa, o fim, contém o término e a aplicação de penalidades. Todas essas etapas compõem o chamado ciclo de vida de um SLA, que será detalhado na próxima seção.

4.2 Ciclo de Vida de um SLA

Em qualquer execução de um sistema existem as fases que marcam o seu início, a sua execução e o seu fim. Dessa forma, o contrato de SLA também possui o seu ciclo de vida, que é o conjunto de fases ou etapas necessários para sua execução em um sistema. O ciclo de vida do SLA pode ser classificado em dois tipos. O primeiro, segundo Sprenkels *et al.* [66] se resume em três fases que são a criação, a operação e a remoção, como pode ser observado na Figura 4.2.

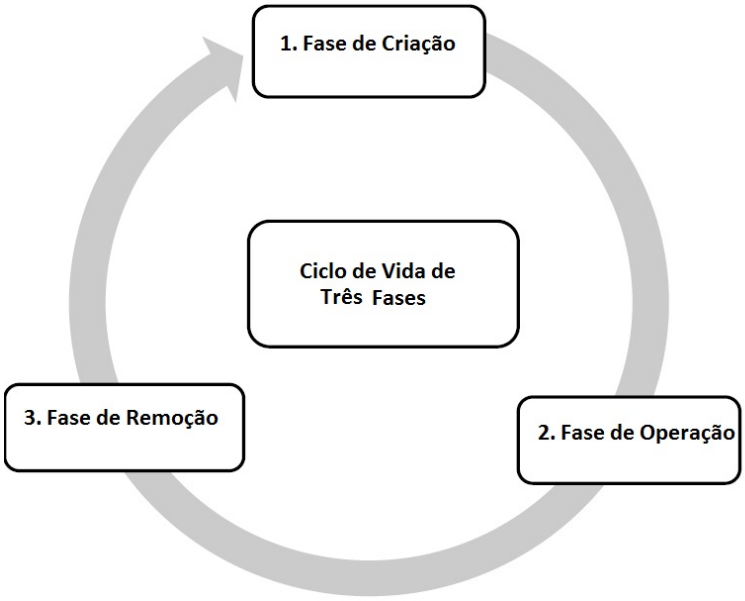

Figura 4.2: Ciclo de Vida de Três Fases de um SLA, adaptado de [90].

O segundo tipo de ciclo de vida do SLA detalha mais a construção e as etapas do SLA. Ele é chamado de ciclo de vida de seis etapas citado por Wu e Buyya [90]. Nesse modelo, o ciclo de vida de um SLA é formado por seis etapas, as quais são: a descoberta, a definição do SLA, o estabelecimento do acordo, o monitoramento de violação de contrato, o término do SLA e a aplicação de penalidades, caso haja violação de contrato, como pode ser observado na Figura 4.3.

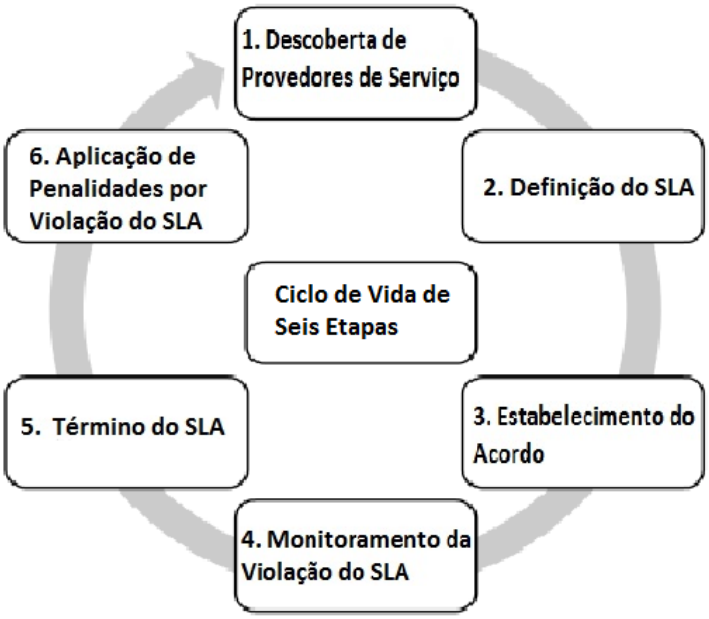

Figura 4.3: Ciclo de Vida de Seis Etapas de um SLA, adaptado de [90].

A partir do ciclo de seis etapas do SLA, é possível analisar cada etapa da seguinte maneira:

- **Etapa 1 - Descoberta**: são descobertos os provedores de serviços que atendem aos requisitos do cliente;
- **Etapa 2 - Definição**: é definido o SLA, que inclui a definição dos serviços, participantes, políticas de penalidades e os parâmetros do *QoS*;
- **Etapa 3 - Estabelecimento de Acordo**: os *templates* de SLA, que são os contratos que identificam os parâmetros do acordo, representando, entre outras coisas, os parâmetros de QoS que um usuário negociou com a arquitetura. Esses contratos são estabelecidos e preenchidos por acordos específicos e os participantes iniciam a consignação do acordo;
- **Etapa 4 - Execução ou Monitoramento**: é realizado o monitoramento da violação do acordo, em que o desempenho da entrega de serviço dos provedores é comparada com o que está no contrato;
- **Etapa 5 - Término**: é encerrado o SLA devido ao tempo de espera, ou a violação por parte dos envolvidos ou término da execução do processamento pedido pelo consumidor;
- **Etapa 6 - Aplicação de Penalidades**: é analisada e aplicada as penalidades correspondentes nas cláusulas do contrato, caso exista violação dos termos de contrato por parte dos envolvidos.

Contudo, há uma relação direta entre o ciclo de vida de três fases e o de seis etapas. Para isso, a fase de criação do ciclo de três fases pode ser mapeado, respectivamente, nas etapas de Descoberta dos provedores de serviços, Definição do SLA e Estabelecimento do Acordo, conforme mostrado na Figura 4.4. A segunda fase, a de Operação, do ciclo de três fases, é mapeada na quarta etapa do ciclo de seis etapas, que é o Monitoramento de Violação. E a última fase, a de Remoção, é mapeada nas etapas de Término do SLA e na Aplicação de Penalidades por violação [10]. Esse mapeamento é claramente mostrado na Figura 4.4.

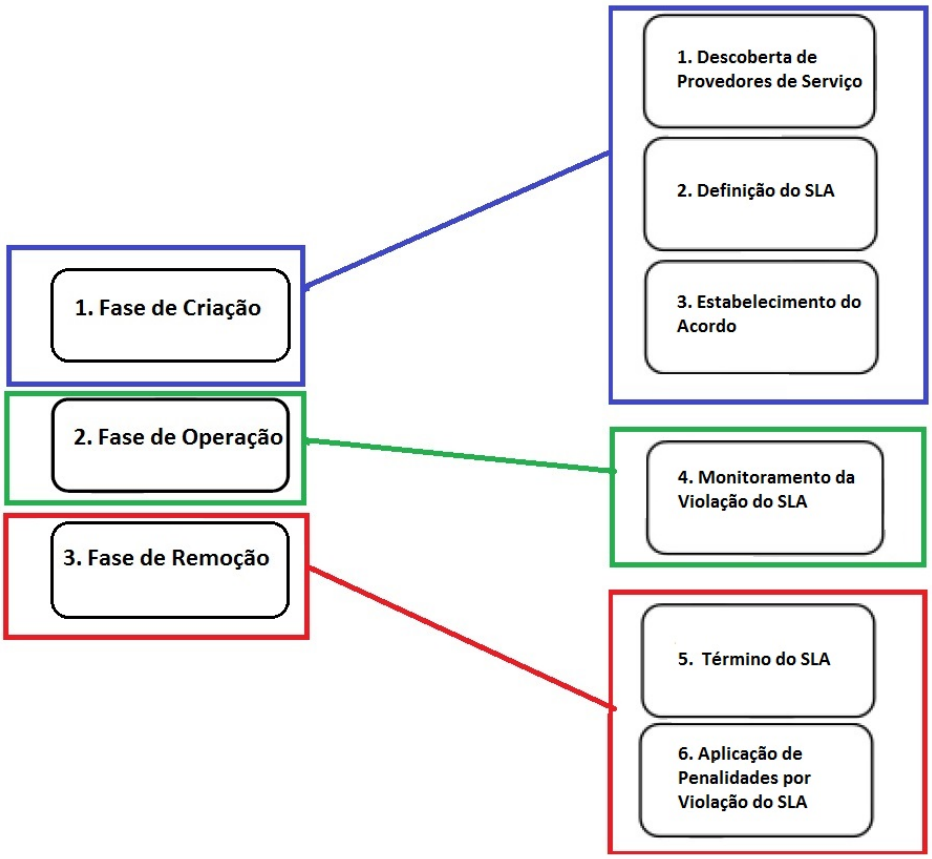

Figura 4.4: Mapeamento do Ciclo de Três Fases no Ciclo de Seis Etapas, adaptado de [90].

Para o gerenciamento desse ciclo existem algumas soluções que utilizam *frameworks* e protocolos para o desenvolvimento de um gerenciador de SLA. A Seção 4.3 apresenta algumas dessas soluções que estão presentes na literatura.

4.3 SLA em Plataformas de Nuvens Federadas

Na literatura há vários trabalhos que tratam de SLA em plataformas de nuvem federadas. No trabalho de Buyya *et al.* [10], por exemplo, a arquitetura em um ambiente computacional de nuvens federadas apresenta um suporte para realizar escalonamento de aplicações por meio de múltiplos provedores de nuvens. Os componentes principais da arquitetura proposta são um negociador de nuvens, uma mediador de nuvens e um coordenador de nuvens. Um cliente inicializa o negociador de nuvens para encontrar a qualidade de serviço desejada, que atendam suas necessidades. O coordenador age como uma porta entre os seus centros de dados internos e nuvens externas, publicando seus serviços para a federação. O mediador de nuvens age realizando acordos juntos com os servidores de serviços e os consumidores. O que implica em uma agregação em demandas

de infraestrutura da aplicação negociadora, e os combinam contra os recursos disponíveis publicados pelos coordenadores de nuvens.

Um outro modelo de SLA é proposto por Weider *et al.* [89], o SLA@SOI, o qual visa introduzir uma solução em sua totalidade de gerenciamento de SLA para serviços orientados a ambientes, que abrange por completo o ciclo de vida do SLA. A arquitetura proposta depende fundamentalmente de dois modelos. Um que descreve os SLAs, permitindo a especificação, tanto em nível funcional quanto em nível não funcional, dos requisitos da Qualidade de Serviço. E o outro que facilita a comunicação entre os componentes envolvidos no gerenciamento do SLA. Dessa forma, são usados diferentes componentes na arquitetura, os principais são [89]:

- A gerência das relações de negócios e políticas;
- A gerência dos modelos, da negociação, do provisionamento e do ajuste do SLA;
- A recuperação de previsões do desempenho de serviço;
- A invocação de implementações de serviços e a coordenação das atividades a serem provisionadas;
- A observação e o monitoramento dos estados dos serviços.

Em Falasi *et al.* [17], o foco é em um modelo de gerenciamento de SLA que busca atender os seguintes objetivos: gerenciar todo o ciclo de vida de um SLA, incluindo definição, negociação, desenvolvimento, monitoramento e aplicação; Refletir a natureza composta do ambiente de nuvens federadas em um gerenciamento multi-nível de SLAs; Ser capaz de esconder a complexidade do gerenciamento de SLA, tanto dos consumidores quanto dos provedores de nuvem; Ser capaz de identificar a origem da interrupção do serviço, causada pela violação do SLA na cadeia de SLAs; e Implementar uma validação dinâmica de SLA, e desenvolver métodos. Além de definir um modelo de gerenciamento de SLA, que administra relações sociais estabelecidas entre diferentes provedores de nuvens.

Em Jrad *et al.* [38] foi designada uma arquitetura genérica de alto nível, integrando vários protótipos de tecnologia de SLA e padrões. O trabalho foi baseado na combinação de várias características de um negociador, os quais foram baseados nos trabalhos de Buyya *et al.* [10], Metsch *et al.* [52] e Kertesz *et al.* [40]. Dessa forma, características como o gerenciamento de SLA, a implementação do serviço, o monitoramento e a seleção do provedor foram incorporados ao trabalho. Além disso, ele é capaz de prover uma camada de abstração que oculta os detalhes técnicos dos provedores de nuvens usando os padrões atuais de nuvens federadas, preparando ainda um *testbed* real para validar e avaliar a arquitetura proposta.

Moustafa *et al.* [58], propõem um *framework* que monitora SLA para serviços em nuvens federadas, que utiliza regras definidas pelos usuários e politícas para mapear os parâmetros do SLA do mais relevante para o menos relevante. Ele utiliza agentes para monitorar métricas especifícas, denominada de SLAM - *SLA Monitoring*, e implementa uma prova de conceito do *framework* sobre o Zabbix 2.2 [47], com JSON-RPC [55], para facilitar a comunicação entre Zabbix e o *framework*.

Para mostrar a diferença entre os trabalhos citados nesta seção, a Tabela 4.1 faz um levantamento sobre as principais características que um controlador de SLA deve levar em consideração. Assim, a primeira coluna da Tabela 4.1 apresenta os principais trabalhos relacionados. A segunda coluna traz os objetivos de nível de serviço, que é definido como o próprio contrato. A terceira coluna apresenta as etapas do ciclo de vida de SLA que são tratadas por cada trabalho. A quarta coluna indica se o trabalho possui mecanismos para auxiliar o usuário do sistema na escolha das máquinas virtuais. A quinta coluna mostra se o trabalho possui código aberto. A sexta coluna apresenta se a segurança está presente ou é tratada por cada trabalho, como cifrar os dados do contrato. A sétima coluna demonstra quais as provedoras de nuvens os trabalhos abordaram. E a oitava e última coluna diz se o trabalho trata de alguma forma a quebra de contrato.

Todavia, como pode ser notado na Tabela 4.1, a maioria dos trabalhos apresentados não aborda todas as etapas do ciclo de vida do SLA, e trabalho [89] que considera todas as etapas, possui código fechado. Além disso, nenhum desses trabalhos considera características específicas das aplicações. Assim, no controlador de SLA proposto neste trabalho, e detalhado no Capítulo 5, considera-se as características das aplicações, em especial, aplicações de Bioinformática. Além disso, o código é aberto e todas as fases do ciclo de SLA são tratadas.

Dessa forma, a ideia do controlador de SLA proposto neste trabalho é garantir a qualidade de serviço para todos os *workflows*, e evitar desperdício de recurso no ambiente federado.

4.4 Considerações Finais

Neste capítulo foram apresentadas as definições de SLA, assim como seus componentes, sua estrutura e seu ciclo de vida. Além disso, também foram apresentados os trabalhos relacionados ao tema, embasando a fundamentação teórica necessária para o entendimento da proposta deste trabalho, que será apresentada no Capítulo 5.

Tabela 4.1: Tabela de Comparação dos Trabalhos Relacionados.

Trabalhos Relacionados	Objetivos de Nível de serviço	Ciclo de Vida	Auxílio de escolha nas máquinas virtuais	Código Aberto	Segurança	Provedoras de Nuvens Envolvidas	Violação do SLA
InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services, Buyya *et al.* [10]	QoS, Desempenho e Disponibilidade	Descoberta, Definição e Monitoramento	Negociador de nuvens(*Cloud Broker*)	Não	Não	Simulador CloudSim	Não deixa claro
Service Level Agreements for Cloud Computing, Weider *et al.* [89]	SLA	Seis Etapas	Sim	Não	Sim	Google, Amazon e Privada	Sim
A Model for Multi-levels SLA Monitoring in Federated Cloud Environment, Falasi *et al.* [17]	SLA	Definição e Monitoramento	Não	Não	Não	Google Cloud	Não se Aplica
SLA based Service Brokering in Intercloud Environments, Jrad *et al.* [38]	SLA	Descoberta, Definição, Estabelecimento de acordo e Monitoramento	Sim	Sim	Não	Simulador CloudSim	Não
SLAM: SLA Monitoring Framework for Federated Cloud Services, Moustafa *et al.* [58]	SLA e Disponibilidade	Descoberta, Definição e Monitoramento	Sim	Não	Não	OpenStack e Amazon EC2	Não

Capítulo 5

Controlador de SLA para Nuvem Federada

Neste capítulo é apresentada a proposta de um controlador de SLA para ambientes de nuvens federadas, utilizando a plataforma de nuvens federadas BioNimbuZ como estudo de caso. Para isso, inicialmente, na Seção 5.1 será apresentada uma visão geral do controlador proposto. Na Seção 5.2 será definida e detalhada a arquitetura do controlador proposto. Em seguida, nessa mesma seção, será detalhado o ciclo de vida implementado pelo controlador de SLA proposto neste trabalho, juntamente com suas funcionalidades. Para finalizar, na Seção 5.3 são apresentados a implementação e a integração do controlador de SLA ao ambiente BioNimbuz, e na Seção 5.4 são feitas algumas considerações finais sobre o controlador proposto.

5.1 Controlador de SLA Proposto

O objetivo do controlador de SLA proposto neste trabalho é que ele seja capaz de implementar, e gerenciar todo o ciclo de vida de cada SLA firmado entre um usuário e uma plataforma de nuvem federada. O Acordo de Nível de Serviço é específico para cada usuário/submissão, assim, para o mesmo usuário pode haver acordos de níveis de serviços diferentes, os quais dependem da aplicação submetida. A ideia é que o controlador de SLA faça toda a gerência de cada ciclo de vida de maneira dinâmica, automática, transparente e simples, de tal maneira que o usuário se preocupe somente com um único contrato por submissão, o qual será entre ele e a plataforma de nuvem federada.

Para isso, o controlador de SLA proposto foi projetado com duas camadas, que são as camadas de Interface e a de Núcleo. A Camada de Interface faz a comunicação com o usuário, para preenchimento dos parâmetros esperados pelo cliente. E a Camada de Nú-

cleo faz a análise, o monitoramento e o controle, relatando para o usuário o monitoramento da execução das tarefas, conforme apresentado na Figura 5.1.

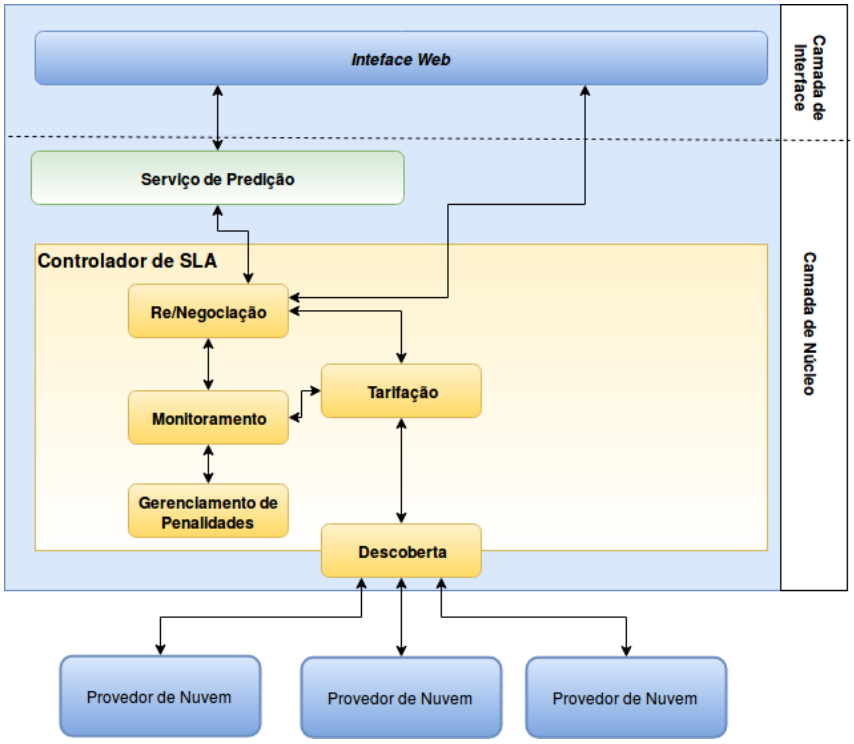

Figura 5.1: Arquitetura do Controlador de SLA para Ambientes de Nuvens Federadas.

Assim sendo, para a implementação de tal controlador, foi necessário desenvolver serviços essenciais para seu funcionamento, tais como serviços de negociação e renegociação, monitoramento, tarifação, descoberta e gerenciamento de penalidades. Esses serviços são detalhados a seguir:

- Negociação e Renegociação: esse serviço tem como objetivo oferecer a opção para o usuário deixar que a plataforma escolha os recursos destinados para a execução do *workflow*; ou que essa escolha seja feita pelo próprio usuário. Esse serviço também é responsável por renegociar um novo contrato, caso haja quebra do contrato em vigor;
- Serviço de Predição: serviço que auxilia o usuário nas escolhas dos recursos mais recomendados para a execução do *workflow*. Esse serviço é uma aplicação isolada que retorna ao controlador as informações de preço, tempo, serviço e tipo de instância por meio de uma mensagem no formato *Json* [15];
- Descoberta: esse serviço está ligado diretamente ao Serviço de Tarifação, pois é o serviço que busca a lista de instâncias disponiveis na federação;

- Tarifação: contabiliza os custos gerados pelo uso dos recursos para execução das tarefas do *workflow*;
- Monitoramento: faz o acompanhamento da execução do *workflow* ao longo de seu ciclo de vida;
- Gerenciamento de Penalidades: serviço responsável por fazer o relatório caso o contrato firmado seja quebrado.

5.2 Arquitetura do Controlador de SLA

A Figura 5.1 apresenta a arquitetura geral do controlador de SLA para nuvens federadas proposto neste trabalho. Como pode ser observado, na camada de Interface há apenas a *interface web* do controlador. Todavia, é na camada de núcleo que todos os serviços do controlador são implementados.

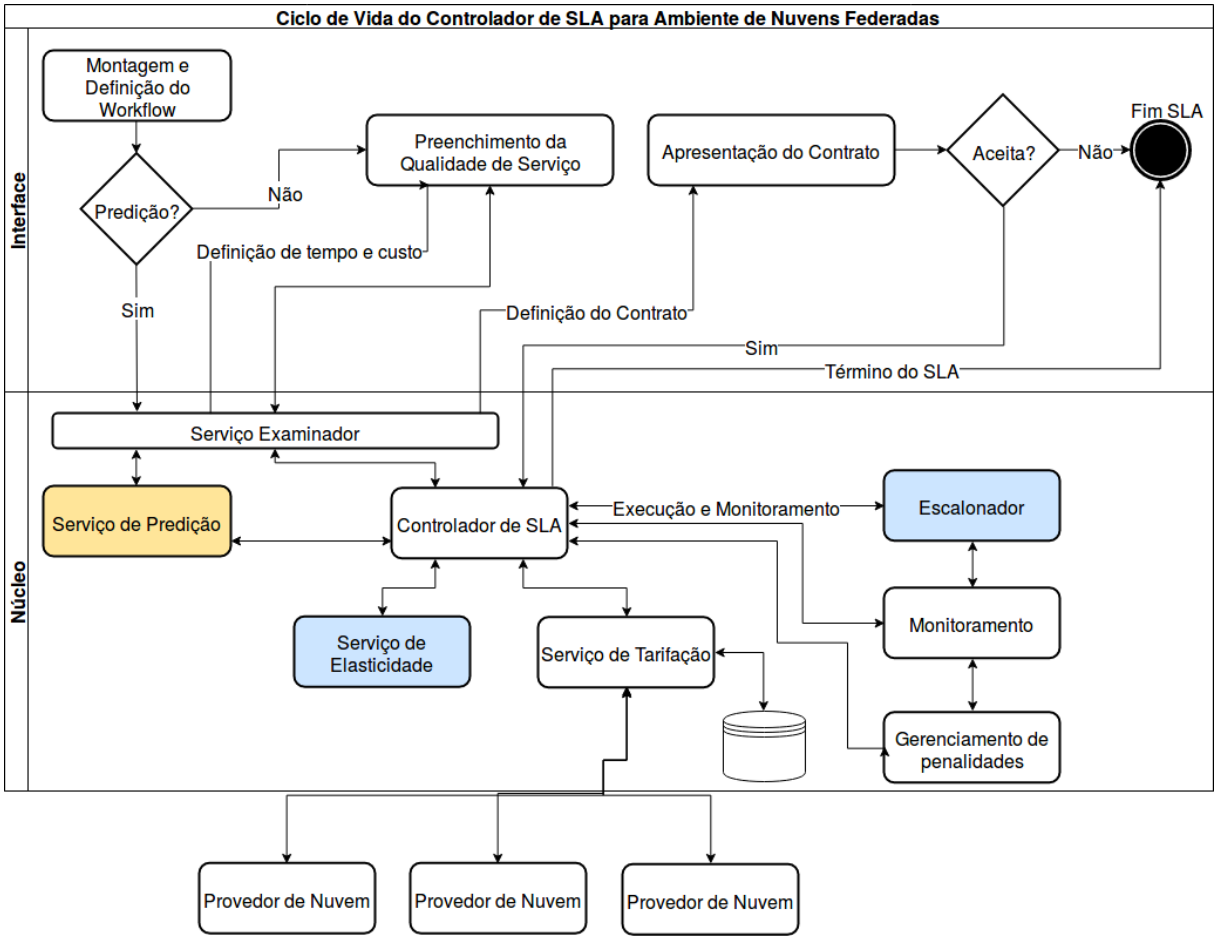

Figura 5.2: Fluxo do Ciclo de vida do SLA para Ambientes de Nuvens Federadas.

Assim, a Figura 5.2 mostra o fluxo a ser seguido para cada acordo de nível de serviço firmado entre os usuários e a plataforma das tarefas submetidas, onde os quadrantes coloridos de azul são os serviços implementados pela plataforma de nuvens federadas. E o quadrante amarelo é uma aplicação isolada e integrada ao controlador de SLA para realizar predição de tarefas para *workflows*. Os demais serviços devem ser implementados diretamente pelo controlador de SLA, fazendo comunicação com a plataforma. Esse fluxo é baseado na proposta de implementar um ciclo de vida de SLA em seis etapas, conforme apresentado no Capítulo 4.

Assim, a Figura 5.3 apresenta o fluxo a ser realizado na etapa 1 do ciclo de vida, que é a descoberta de provedores (ou recursos) que atendam aos parâmetros de qualidade de serviço. Os passos a serem realizados na etapa 1 são:

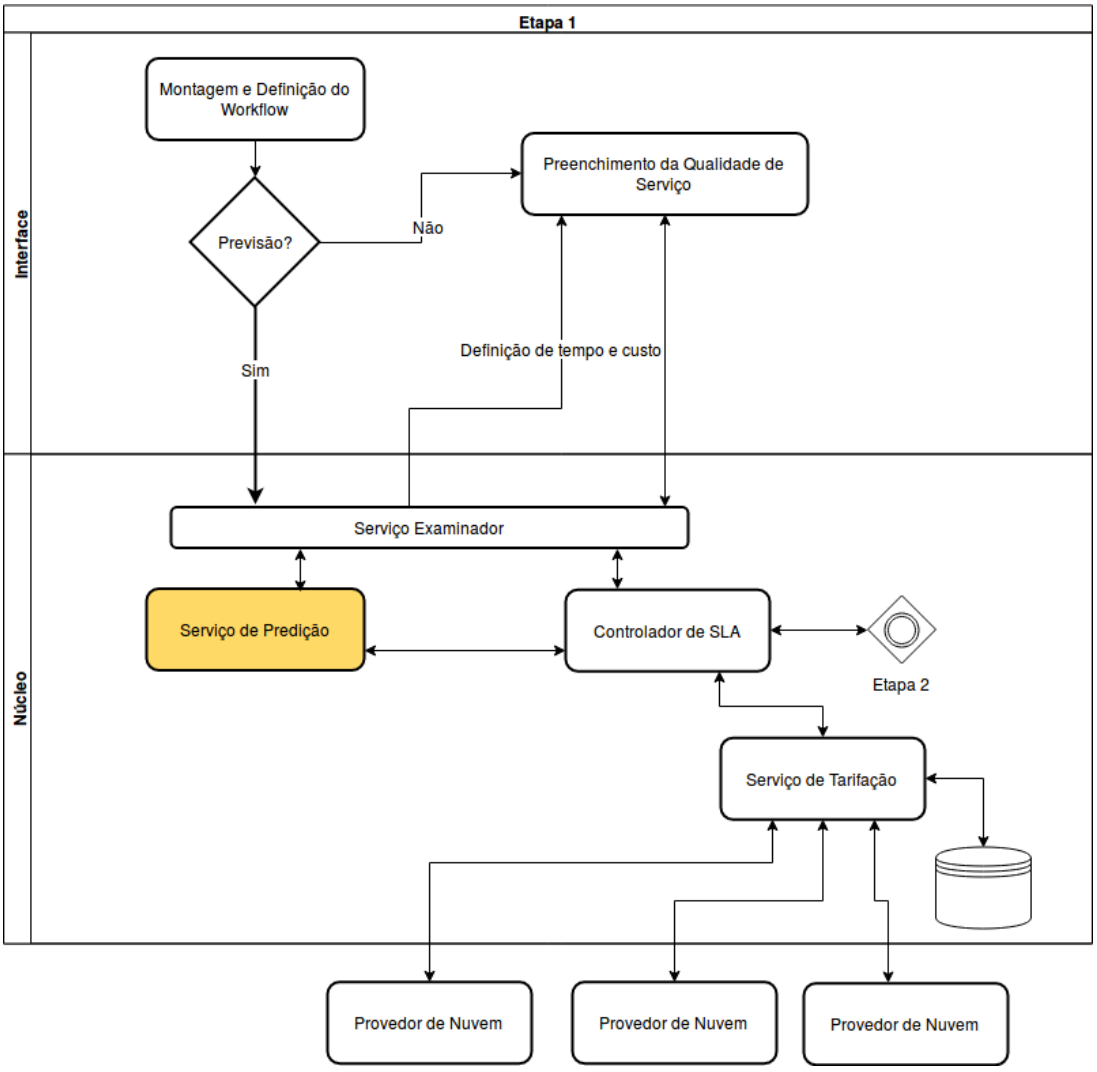

Figura 5.3: Etapa 1 do Ciclo - Descoberta de Provedores.

- Passo 1 - o usuário, por meio da *interface web*, monta o *workflow* informando: quais

tarefas ele irá executar e quais são os arquivos de entrada para cada tarefa. Após essa montagem, o usuário decide se quer utilizar o serviço de predição, que é uma aplicação isolada e integrada ao controlador de SLA, para realizar predição de tarefas para *workflows*. Com o preenchimento dessas informações envia-se uma requisição para o Núcleo no qual o Controlador de SLA (ou o Serviço de Predição) faz o papel de *Serviço Examinador* de requisição, para que a proposta seja atendida;

- Passo 2 - o *Serviço Examinador*, que é exercido pelo serviço de predição ou pelo controlador de SLA, lê os parâmetros passados. Caso o usuário tenha escolhido utilizar o serviço de predição, esse irá fazer o papel do serviço examinador, analisando os parâmetros do *workflow* e contactando o controlador de SLA. Esse por sua vez vai disparar o serviço de tarifação, que faz o papel do serviço de descoberta, buscando a lista de máquinas disponíveis na federação, e retorna para o serviço de predição. Com essa lista de recursos disponíveis e uma base histórica de execuções, que o serviço de predição gera, ele processa as informações, objetivando minimizar o custo e o tempo de execução. Em seguida, a lista das soluções encontradas, as estimativas de tempo e o custo da execução são enviados para o controlador de SLA.

  Caso o usuário não escolha utilizar o serviço de predição, é enviado uma solicitação ao controlador de SLA, que por sua vez manda uma requisição para o serviço de tarifação, solicitando a lista de tipos de instâncias disponíveis, retornando assim a lista das soluções para a interface.

- Passo 3 - o controlador de SLA passa a lista de recursos disponíveis com seus respectivos custos à interface, a fim de definir o contrato a ser firmado.

Em seguida, é iniciada a etapa 2 das 6 etapas do ciclo de vida do SLA, que é a definição do SLA. Logo, para o cumprimento da etapa 2 são realizados os passos mostrados na Figura 5.4, e explicado a seguir:

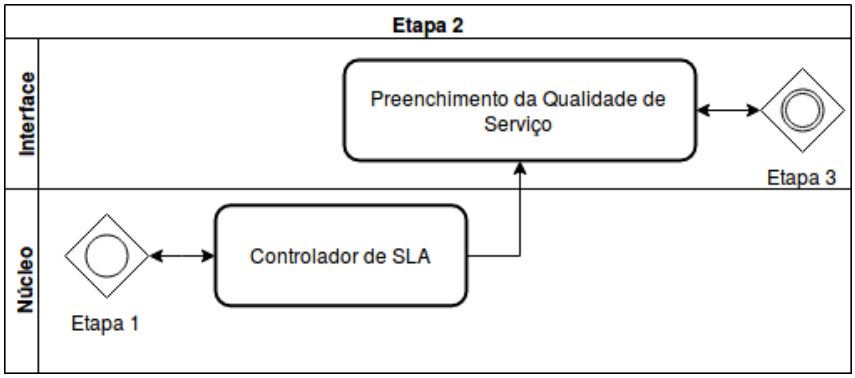

Figura 5.4: Etapa 2 do Ciclo - Definição do SLA.

- Passo 4 - na interface do usuário são apresentados os recursos selecionados. Assim, caso o usuário aceite as escolhas dos recursos para a execução do *workflow*, feita pelo serviço de predição, será apresentado o contrato com as qualidades de serviços escolhidas e seus componentes. Nesse caso, o SLA monitora cada recurso escolhido para executar uma tarefa específica, como seu tempo e seu custo, proposto pelo serviço de predição.

  Por outro lado, caso o usuário não aceite os recursos escolhidos pela predição, é apresentada uma lista completa de todos os recursos disponíveis para o usuário fazer suas escolhas, com um limite de 20 instâncias por provedor. Esse limite é padrão pelos provedores, podendo o usuário optar por limitar o tempo ou o custo em que deseja que seus recursos fiquem provisionados para a execução das tarefas. Além disso, é importante preencher o quanto o usuário está disposto a pagar a mais, caso seja atingida a limitação imposta, e a execução não tenha terminado.

  Completando essas informações, é construído o contrato com os componentes do SLA que são:

  - **Propósito**: que constitui os objetivos a serem alcançados no uso do SLA, os quais seriam maior desempenho, menor custo ou uma média entre desempenho e custo;
  - **Restrições**: que são passos ou ações necessárias de serem tomadas para garantir que o nível de qualidade de serviço seja fornecido, o qual será o monitoramento parcial da execução do *workflow*;
  - **Período de Validade**: o tempo em que o SLA será válido, dependendo das limitações impostas ou as definições previstas pelo serviço de predição;
  - **Escopo**: que descreve as tarefas que serão entregues e cobertas pelo SLA, no qual estão as tarefas do *workflow* escolhidas;
  - **Participantes do Contrato**: que são o cliente e a plataforma de nuvem federada envolvida;
  - **Objetivos de Nível de Serviço (***Service Level Objectives - SLO's***)**: em que as partes do contrato devem concordar com os parâmetros de qualidade de serviço, tais como disponibilidade, desempenho e confiabilidade;
  - **Penalidades**: penalidades aplicadas, por meio de créditos, nos casos em que os serviços ofertados não sejam cumpridos de acordo com os objetivos de nível de serviço. A penalidade por meio de créditos foi escolhida pela maior adesão no mercado de computação em nuvem;

– **Serviços Opcionais**: que podem ser necessários, como por exemplo transferência de dados;

- Passo 6 - Após o contrato ter sido definido, o mesmo é apresentado na *Interface Web* junto com suas cláusulas.

Dando continuidade ao ciclo de vida de um SLA, na etapa 3 do ciclo tem-se o estabelecimento de acordo entre as partes, que envolve o passo 7, abordado na Figura 5.5, e descrito como:

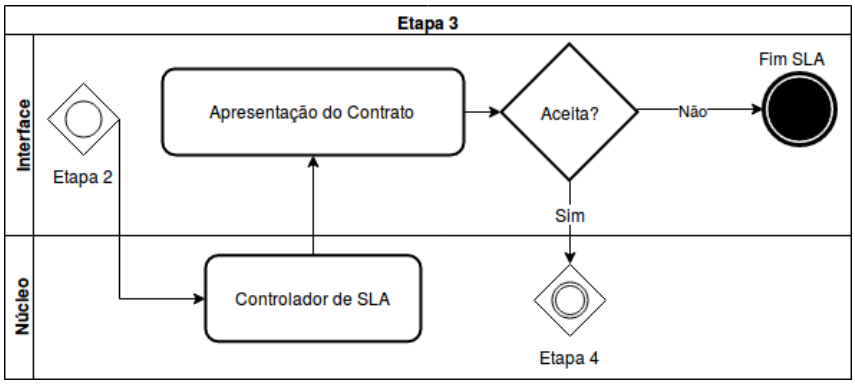

Figura 5.5: Etapa 3 do Ciclo - Estabelecimento de Acordo.

- Passo 7 - com a definição do SLA na etapa 2 do ciclo de vida, o SLA é criado para o usuário. Nesta etapa o usuário tem a opção de escolher ou declinar o acordo. Caso o usuário aceite a proposta, a aplicação envia o SLA para o controlador. Caso contrário, o contrato acaba nesta etapa.

Com o estabelecimento do acordo entre as partes na etapa 3, inicia-se a etapa 4 do ciclo de vida do SLA, que é a execução do SLA e o monitoramento, que contém os passos apresentados na Figura 5.6, e descritos a seguir:

- Passo 8 - com o SLA escolhido, o controlador de SLA inicia sua operação, sinalizando para o serviço de elasticidade criar as instâncias escolhidas em seus respectivos provedores. Após a criação das instâncias, o controlador de SLA envia a requisição do *workflow* para o Serviço de Escalonamento, o qual começa a execução. Enquanto isso, o controlador de SLA monitora a execução de forma constante. Nesse monitoramento são realizados cálculos parciais de custos baseados na estimativa do custo calculado pelo serviço de predição, caso o usuário tenha escolhido essa opção; ou pela limitação de custo ou tempo, caso o usuário não tenha escolhido a predição. Os cálculos realizados pelo controlador de SLA proposto são descritos na Seção 5.2.1

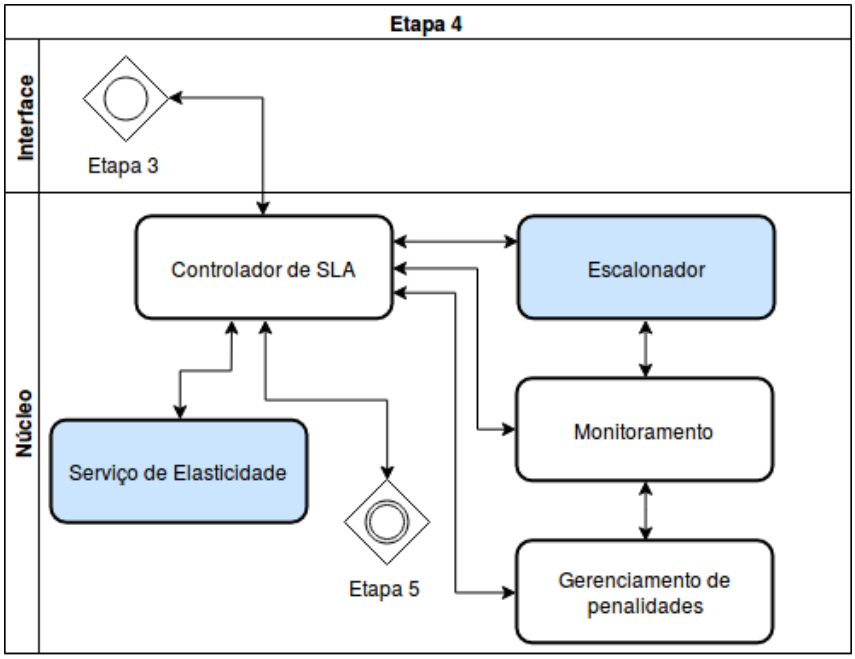

Figura 5.6: Etapa 4 do Ciclo - Execução e Monitoramento do SLA.

5.2.1 Cálculos do Controlador de SLA

Para que o controlador de SLA possa fazer a verificação parcial do cumprimento (ou não) do contrato, faz-se necessário calcular os tempos e os gastos fornecidos pelo serviço de predição. Para isso, quando o usuário aceita a predição feita, este serviço retorna uma lista com os tempos e os custos associados a cada tarefa, incluindo também o tipo de instância escolhida para determinada tarefa. Desse modo, para calcular o custo total do *workflow* é necessário percorrer a lista fazendo um somatório do tempo de cada tarefa, multiplicado pelo custo da instância, resultando assim na equação representada pela Fórmula 5.1:

$$CustoTotalWorkflow = \sum_{k=1}^{N} k; \tag{5.1}$$

Na Fórmula 5.1, a variável k é dada por $TempoServico * CustoPorHoraInstancia$. A variável $TempoServico$ representa o tempo estimado pelo serviço de predição para executar determinada tarefa. A variável $CustoPorHoraInstancia$ representa o custo por hora do tipo de instância ou recurso que o serviço de predição escolheu, e o custo total do *workflow* é simbolizado pela variável $CustoTotalWorkflow$ que é denotado pelo somatório de k.

Com o custo total do *workflow* calculado, e com as informações de tempo e de custos de cada tarefa, tem-se a verificação das tarefas que compõem o *workflow*, no qual é monitorado se o tempo e o custo das tarefas previstas foram alcançadas, informando para o usuário caso tenha sido alcançado. Essa verificação é dada pelas Fórmulas 5.2 e 5.3.

$$(TempoAtual - TempodeCriacaoInstancia) > TempoServicoPredicao \quad (5.2)$$

Na Fórmula 5.2 a variável $TempoAtual$ representa o tempo exato do instante em que está sendo verificado. A variável $TempodeCriacaoInstancia$ é a representação do exato momento em que a instância foi iniciada, e a operação de subtração entre essas duas variáveis representa o *uptime*, tempo em que o recurso está disponível. Isso verifica se o *uptime* da tarefa é superior ao tempo previsto, pelo serviço de predição, para aquela tarefa na qual é representada pela variável $TempoServicoPredicao$.

E para a verificação, se o custo das tarefas atingiram o custo estimado para determinada tarefa, tem-se a Fórmula 5.3:

$$uptime * CustoPorHoraInstancia > CustoServicoPredicao \quad (5.3)$$

Na qual a variável *uptime* representada pela Fórmula 5.4, significa o tempo atual menos o tempo em que a instância foi criada. A variável $CustoPorHoraInstancia$ é o valor por hora da instância escolhida para executar tal tarefa, e o custo da tarefa previsto pelo serviço de predição, verificando assim, se o sistema atingiu o custo previsto.

$$uptime = TempoAtual - TempodeCriacaoInstancia \quad (5.4)$$

Essas verificações e cálculos parciais apenas disparam mensagens de alertas para o controlador de SLA, e informam para o usuário a situação da execução. Todavia, para a remoção das instâncias, outras verificações e cálculos são realizados. Somente quando o custo total do *workflow* for atingido, calculado tanto pela previsão do serviço de predição ou pela a limitação de custo, essas instâncias são removidas.

Para remover as instâncias/recursos utilizados para a execução do *workflow*, são realizados os cálculos definidos pelas Fórmulas 5.5 e 5.7. Ao fato que, o limite definido pelo o usuário ou o custo total previsto pelo serviço de predição sejam

atingidos, a Fórmula 5.5 calcula esses valores, para assim disparar um evento em que serão encerrados os recursos alocados para a execução daquele *workflow* específico.

$$limitCost! = null\ and\ currentCost > toleranceCost + limitCost \tag{5.5}$$

A Fórmula 5.5 verifica se foi atribuído para o *workflow* um limite de custo, esse limite de custo é representado pela variável *limitCost*. Essa variável possui valores diferentes para os seguintes casos: 1 - no primeiro caso ela é o custo total do *workflow*, quando o usuário optar pelo serviço de predição; 2 - no segundo caso ela é o valor da limitação de custo em que as instâncias devem ficar ativas, quando o usuário não optar pela predição e impor um limite de custo para essa execução; verifica se o custo atual, representado pela variável *currentCost*, e simbolizado pela Fórmula 5.6, é maior que o custo total do *workflow*. Este custo total é simbolizado pela soma do limite de custo, representado pela variável *limitCost*, e do custo de tolerância representado pela variável *toleranceCost*, que é uma margem de despesa preenchida pelo usuário, caso o *workflow* não tenha terminado, finalizando assim os recursos caso esse limite seja alcançado.

$$currentCost = \sum_{w=1}^{N} w \tag{5.6}$$

Como visto na Fórmula 5.6, no qual w é dado por $(uptime + Variancia) * CustoPorHoraInstancia$, o *uptime* que é definido pela Fórmula 5.4, a variável $Variancia$ é definida pelo *delay* em que o controlador faz o monitoramento dos SLAs. Logo, o custo atual do *workflow* é definido pelo somatório do *uptime* mais o *delay* do monitoramento do controlador de SLA multiplicado pelo custo da instância, sendo representado pela variável *currentCost*.

Assim, nos casos em que o usuário tenha escolhido limitar o tempo em que as instâncias fiquem ativas, os cálculos necessários para a realizar essa verificação é definido pela Fórmula 5.7.

$$TempoAtual - period > limitTime - Variancia \tag{5.7}$$

Na Fórmula 5.7 é feita a verificação se o tempo atual menos o horário da criação da primeira instância, representada por *period*, é maior que o limite de tempo imposto pelo usuário, definido por $limitTime$, menos o *delay* do monitoramento do controlador de SLA, representado pela variável $Variancia$. Caso essa verificação

seja verdadeira, é analisado se o usuário está disposto a pagar uma quantidade a mais, o qual é dado pela Fórmula 5.8

$$TempoAtual - period > toleranceTime + limitTime - Variancia \tag{5.8}$$

Na Fórmula 5.8 o horário da verificação é representado pela variável $TempoAtual$. O horário da criação da primeira instância do *workflow* é representado pela variável *period*. A variável $toleranceTime$ representa o tempo do valor em que o usuário está disposto a pagar, caso o *workflow* não termine no tempo, definido pela Fórmula 5.9. O tempo limite imposto pelo usuário é simbolizado por $limitTime$ e o *delay* do monitoramento do controlador de SLA, representado pela variável $Variancia$, ou seja, a equação da verificação é dada pelo tempo atual, menos o horário de criação da primeira instância do *workflow*, se é maior do que o tempo do valor que o usuário está disposto a pagar, mais o tempo estipulado para que as instâncias do *workflow* fiquem ativas, menos o *delay* do controlador de SLA. Para transformar o custo a mais em que o usuário está disposto a pagar, a Fórmula 5.9 realiza o cálculo desse valor.

$$toleranceTime = (toleranceCost/workflowCostPerHour{*}ONEHOURMILLES) \tag{5.9}$$

Onde $toleranceTime$ é a representação do custo de tolerância, ou seja, é a representação em tempo, do valor em que o usuário está disposto a pagar a mais, caso o *workflow* não tenha terminado. Essa representação do custo de tolerância é calculado pela divisão entre esse custo a mais que usuário está disposto a pagar, $toleranceCost$, e o custo por hora do *workflow*, $workflowCostPerHour$ variável essa que é calculada pela Fórmula 5.10, multiplicado pelo valor de uma hora em milissegundos, $ONEHOURMILLES$.

$$workflowCostPerHour = \sum_{r=1}^{N} r \tag{5.10}$$

Assim, na Equação 5.10 o r é dado por $CustoPorHoraInstancia$, que é a representação do custo de cada instância alocada para a execução do *workflow*. Além de checar as informações das instâncias criadas pelos provedores, tais como se os mesmos estão disponibilizando aquelas configurações escolhidas pelos tipos de instâncias ofertadas, que foi realizada na etapa 1. Caso essas parciais não estejam sendo cumpridas de acordo com o contrato firmado, sinaliza-se para uma base de

informações lógicas da plataforma e para o usuário que há uma anomalia na execução. Para os casos em que a soma dessas parciais na execução do *worflow* for maior do que o acordado, serão aplicadas as penalidades cabíveis em forma de créditos concedidos, conforme descrito no SLA.

Após o início da etapa 4 do ciclo de vida do SLA, a qualquer momento pode ocorrer a etapa 5 do ciclo, que é o término do SLA. E a etapa 6, que é a aplicação de penalidades por violação de contrato. As duas etapas estão contidas nos passos apresentados na Figura 5.7, e descritos nos passos a seguir:

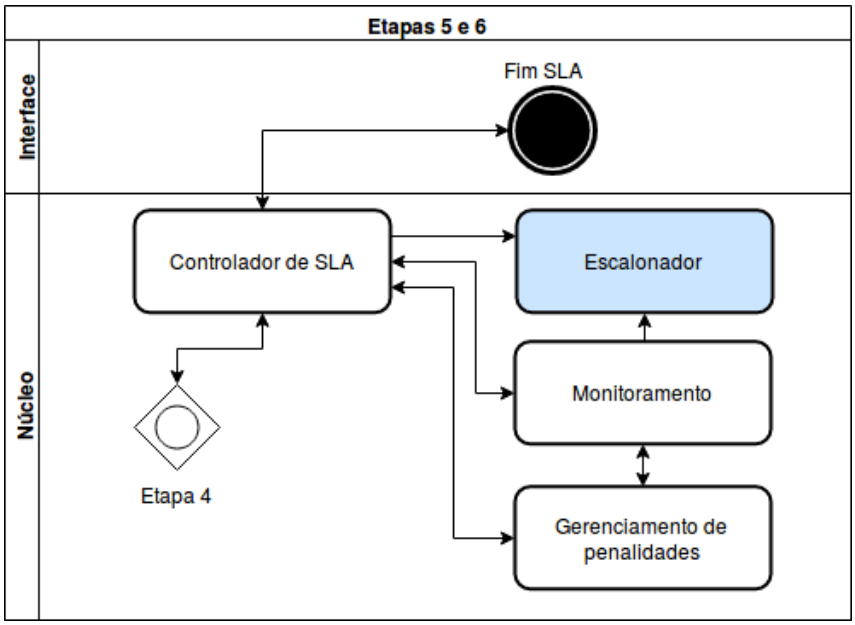

Figura 5.7: Etapas 5 e 6 do Ciclo - Término do SLA e Aplicação de Penalidades.

- Passo 9 - o término do SLA poderá ser causado por algumas ações, as principais são:
  - Pelo Usuário: pode ocorrer se o usuário cancelar a execução quando ele desejar, pagando assim pelo que tiver usado; Pela limitação de tempo que o usuário condicionou na simulação de custo; Ou pela limitação de custo que o usuário configurou na simulação;
  - Pelo Sistema: por quebra de contrato a partir do usuário, caso em que o usuário não pagou sua fatura anteriormente (caso utilize uma federação que não cobre para utilização dos recursos, não será faturado, mas caso sejam utilizadas nuvens públicas o serviço de tarifação irá fazer a contabilidade da fatura); ou por parte do sistema quando o mesmo fica indisponível no meio da execução do *workflow*. Nesse caso, ocorre as penalidades descritas no contrato, que será concessão de créditos para o cliente.

– Por Finalização da Execução: em que o sistema finaliza a execução do *workflow* de forma normal dentro do esperado.

Com o término do SLA ou da execução do *workflow*, a sexta e última etapa do ciclo de vida, é iniciado pelo controlador de SLA, que é a aplicação de penalidades por violação de contrato. Nessa etapa, o controlador de SLA verificará, na base de informações lógicas da plataforma, se a soma total das parciais de tempo ultrapassou as do contrato, ou se a execução foi terminada pelo sistema de forma correta, ou se houve quebra de contrato por parte do sistema ou do usuário. Assim, são concedidos créditos para o usuário, caso tenha sido falha do sistema, ou caso o usuário tenha finalizado a execução antes do que estava descrito no acordo, será cobrado a hora cheia pelo uso do sistema.

Dessa forma, o controlador de SLA para uma plataforma de nuvem federada, proposto neste trabalho, encerra o ciclo de vida do SLA, abordando todas as etapas presentes no ciclo de seis etapas citado por Buyya *et al.* [90].

5.3 Integração com a Plataforma BioNimbuZ

Para testar o controlador de SLA proposto neste trabalho, foi escolhida a plataforma de nuvens federadas BioNimbuZ. Para isso, alguns ajustes foram feitos na atual arquitetura da plataforma BioNimbuZ para receber o controlador de SLA. Os principais ajustes foram realizados na interface web, no Serviço de Tarifação, no Serviço de Elasticidade, no Serviço de Monitoramento, no Controlador de Usuário e no Serviço de Escalonamento. Todas as mudanças serão explicadas nas seções seguintes.

5.3.1 Interface web

Para o preenchimento das informações necessárias ao contrato foram criadas telas na interface web, contendo questionamentos e opções para a obtenção dessas informações. O objetivo desta mudança foi garantir o preenchimento dos itens de qualidade a serem cumpridos no contrato. Esses itens são, então, passados para o controlador de SLA por meio da Camada de Integração do BioNimbuZ, que utiliza o *framework resteasy* [69], conforme apresentado na Figura 5.8. Conforme a tela apresentada na Figura 5.8, o usuário já optou pela escolha dos recursos e não optou pela opção do serviço de predição, escolhendo assim uma máquina que servirá para a execução do *workflow* por ele criado.

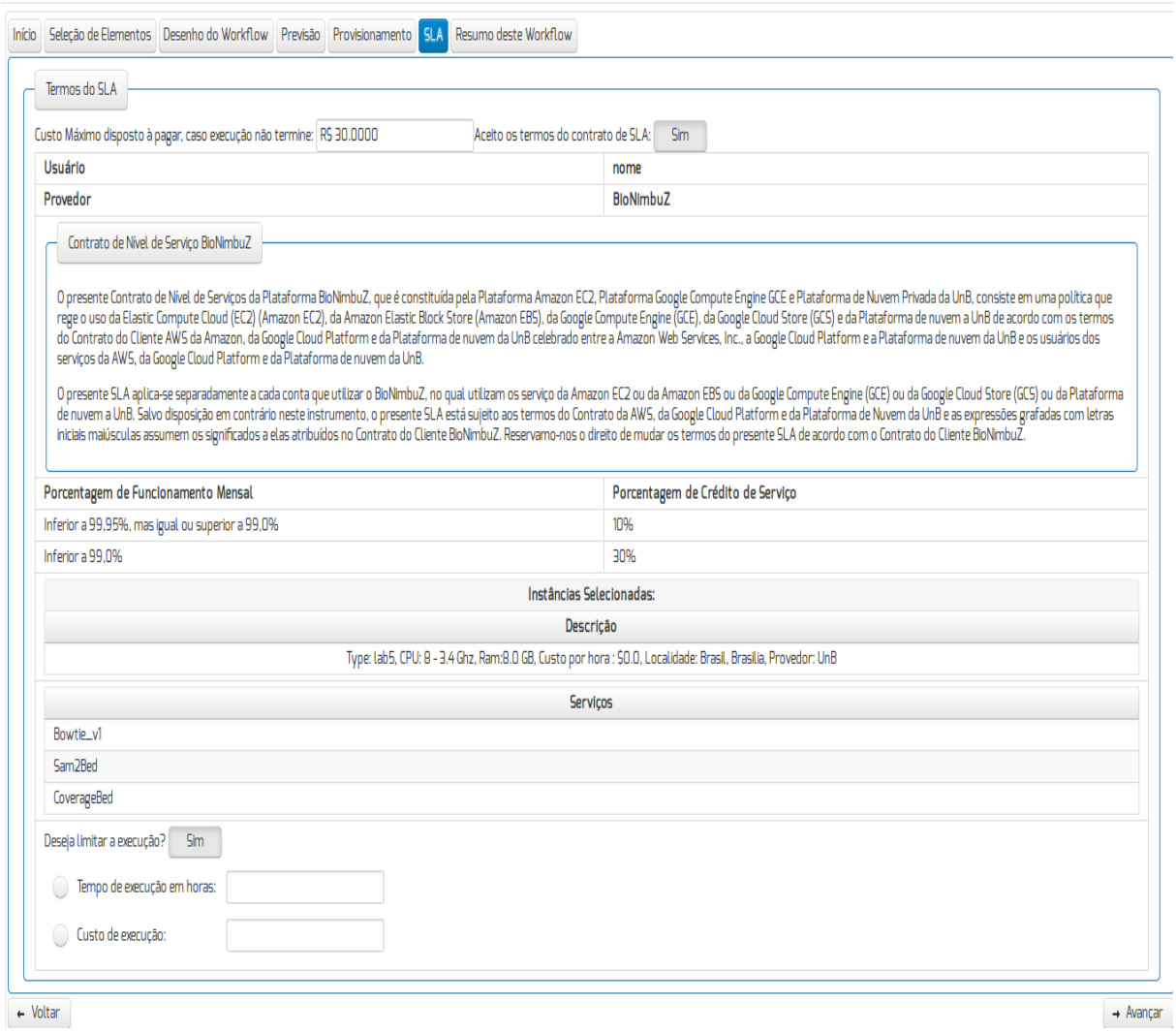

Figura 5.8: Página do BioNimbuZ com o Contrato de SLA.

5.3.2 Serviço de Tarifação

O Serviço de Tarifação da plataforma BioNimbuZ teve que ser adaptado para que o mesmo buscasse a informação de todos os tipos de instâncias disponíveis nas nuvens integrantes da federação, tais como na Amazon EC2 [78], na Google Cloud Compute Engine [33], e as máquinas físicas disponibilizadas pela nuvem privada do Laboratório de Bioinformática e Dados - LABID, retornando então uma lista de instâncias para a aplicação. A Figura 5.9 apresenta um exemplo dessa lista.

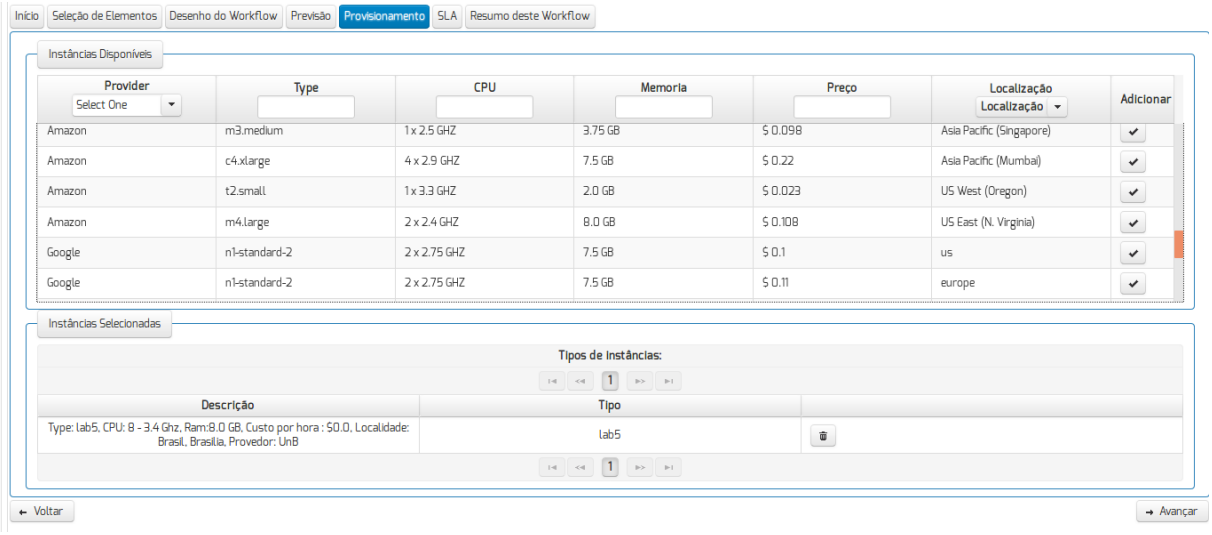

Figura 5.9: Lista com as Instâncias Fornecidas pela Plataforma.

O Serviço de Tarifação possui um mecanismo que busca em nuvens públicas, tais como Amazon EC2 [78] e Google Cloud Compute Engine [33], um JSON [15] com as informações das instâncias. Essas informações contém as configurações e o preço por hora da instância, que é disponibilizado pelos provedores de forma pública. Essa busca ocorre periodicamente em um intervalo de 24 horas, atualizando a lista dessas instâncias de forma em que, caso ocorra alteração de preço, esse preço seja atualizado. Isso auxilia o usuário da plataforma a escolher a(s) instância(s) que ele deseja utilizar em seu *workflow*.

5.3.3 Serviço de Elasticidade

O Serviço de Elasticidade foi alterado para realizar a criação e a remoção das máquinas virtuais de forma automática, aceitando os tipos de instâncias ofertados pelos provedores de nuvem. Para isso, foram utilizadas as *APIs aws-java-sdk* e *google-api-services-compute*, ambas da Amazon [79] e da Google [65], respectivamente. Essas APIs implementam métodos comuns, tais como a criação e a remoção de instâncias independente do provedor.

No método de criação são utilizados os seguintes parâmetros: provedor de nuvem, onde a instância será criada; o tipo da instância que contém as configurações da mesma, juntamente com o seu preço; e o nome da instância para melhor diferenciar o nome das instâncias para cada *workflow*.

No método de remoção é passado o nome do provedor em que a instância foi criada, e o IP da instância correspondente. Facilitando, assim, a criação, o monitoramento e a remoção dessas instâncias para o controlador de SLA, em que o mesmo envia essas requisições para o Serviço de Elasticidade.

5.3.4 Serviço de Monitoramento

No Serviço de Monitoramento foi adicionado mais um parâmetro a ser analisado. Esse parâmetro contém as informações do usuário, as instâncias e os contratos para cada *workflow*. Essa mudança implicou na necessidade de adicionar um nó na estrutura lógica mantida pelo Zookeeper na plataforma BioNimbuZ, conforme é apresentado na Figura 5.10

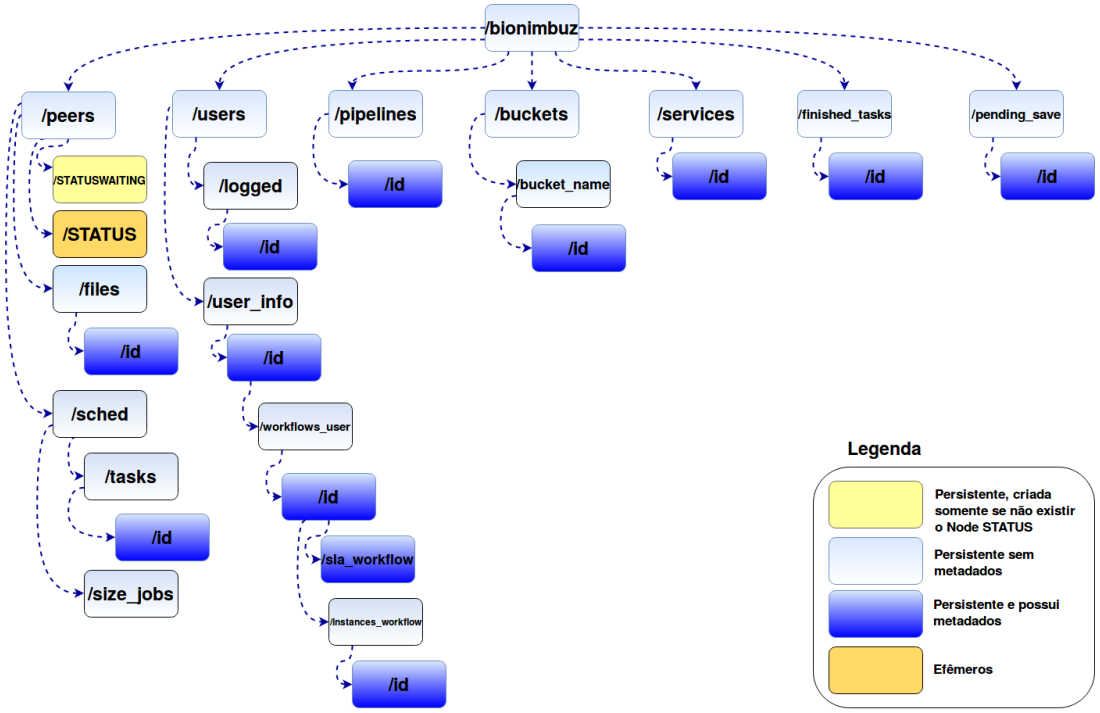

Figura 5.10: Estrutura Hierárquica dos *Znodes* no BioNimbuz [68].

Essa adaptação permitiu que a lista de usuários com seus respectivos *workflows* e SLAs, esteja disponível para toda a plataforma. Assim, é possível monitorar individualmente cada *workflow* e atualizar seu *status*. Isso garante que toda a plataforma saiba das informações a respeito dos *workflows* de cada usuário.

5.3.5 Controlador de Usuário

O Controlador de Usuário foi adaptado para poder adicionar as informações do contrato de SLA a cada *workflow* do usuário, no momento em que for registrado o *workflow* para o usuário. Dessa forma, este serviço é o responsável por fornecer as informações de cada usuário para a plataforma, fazendo comunicação com o Controlador de *Jobs*, o Controlador de SLA e o Serviço de Monitoramento.

Essa mudança foi essencial para o monitoramento dos SLAs de cada usuário, pois estas informações são necessárias para a comparação e verificação dos cálculos do controlador de SLA.

5.3.6 Serviço de Escalonamento

O Serviço de Escalonamento teve que ser adaptado para diferenciar os recursos atribuídos de cada usuário para as tarefas específicas de cada *workflow*. Essa alteração ocorreu por meio da inserção do endereço da instância, caso o usuário tenha escolhido utilizar o Serviço de Predição; ou uma lista de endereços, caso o usuário deseje escolher as configurações das instâncias por ele mesmo, em cada tarefa que compõem o *workflow* montado pelo o usuário.

Dessa forma, o escalonador passou a ter a função de analisar o *workflow*, particioná-los em tarefas e associar essas tarefas aos recursos que o usuário disponbilizou, associando assim as tarefas à endereços das máquinas virtuais criadas para cada usuário.

5.4 Considerações Finais

Neste capítulo foi apresentado o Controlador de SLA proposto neste trabalho, e implementado utilizando a plataforma BioNimbuZ, como estudo de caso. O Controlador de SLA irá mudar a forma de tratamento dos Acordos de Níveis de Serviço no BioNimbuZ, pois os contratos se tornarão mais flexíveis, transparentes e ágeis. Para comprovar isso, o Capítulo 6 apresenta os resultados obtidos a partir dos testes realizados com *workflows* reais de Bioinformática.

Capítulo 6

Estudo de Caso

Neste capítulo é descrito o estudo de caso realizado com o controlador de SLA, utilizando a plataforma BioNimbuZ. Na Seção 6.1 é descrito o ambiente montado para a realização dos testes e a sequência de execução para os testes. Na Seção 6.2 são demonstrados os relatórios gerados para o usuário, caso as máquinas não atendam a configurações e/ou os limites impostos pelo usuário sejam atingidos. Na Seção 6.3 é apresentada uma análise sobre o *overhead* adicionado à execução do *workflow*, com a inserção do controlador de SLA. E na Seção 6.4 é apresentada a economia que o usuário poderá ter utilizando o controlador de SLA proposto. Para finalizar, a Seção 6.5 faz um resumo sobre os itens tratados neste capítulo.

6.1 Ambiente de Testes

Para a realização dos testes foram utilizadas duas máquinas virtuais na Google, ambas com as configurações de 2 núcleos e 7,5 GB de memória RAM. Uma dessas máquinas foi usada como servidor *web* e a outra como núcleo do BioNimbuZ. Além dessas máquinas bases, foram montados três ambientes, nos quais foram executados o *workflow*. O primeiro ambiente foi composto por duas máquinas de 2 CPUs e 2 GB de RAM. O segundo ambiente foi composto por duas máquinas com 4 CPUs, 8 GB de RAM. Por fim, o terceiro e último ambiente foi composto por duas máquinas de 8 CPUs e 8 GB de RAM, cada um desses ambientes teve uma máquina na Amazon e outra na Google.

Além disso, cada um desses ambientes executou cinco vezes o *workflow*, apresentado na Figura 6.1. Esse *workflow* tem como objetivo identificar genes diferencialmente expressos em células humanas cancerosas do rim e do fígado [49] [71], com fragmentos gerados pelo sequenciador Illumina [35]. O *workflow* consiste em quatro etapas, as quais são apresentadas na Figura 6.1.

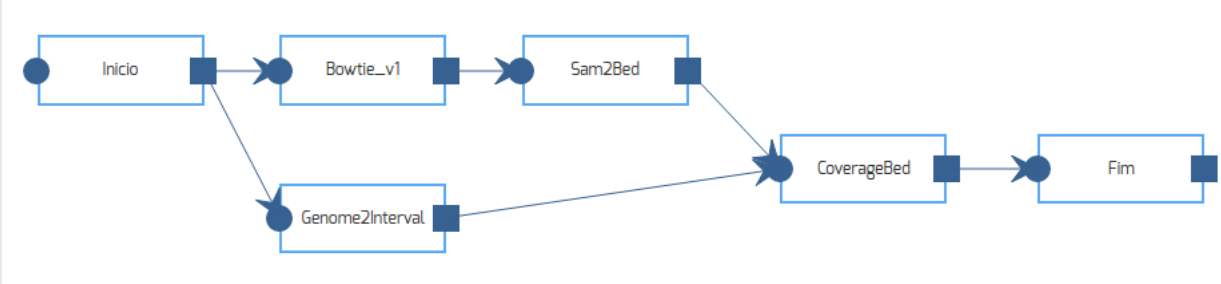

Figura 6.1: *Workflow* Utilizado nos Testes.

Na primeira etapa a ferramenta **Bowtie** [43] é utilizada para mapear os fragmentos nos 24 cromossomos humanos de referência, isto é, o software identifica a região do genoma de referência de onde cada fragmento de entrada está localizado. Assim, quando um conjunto de fragmentos é mapeado na mesma região, pode-se inferir que possuem a mesma organização estrutural do genoma de referência.

Na segunda etapa é realizada uma conversão de tipos de dados, na qual o arquivo de saída do Bowtie (arquivo .SAM) é convertido para o formato .BED. Para isso, é utilizado um *script* de conversão, chamado ***sam2bed*** [59].

Dessas quatro etapas, a terceira etapa pode ser feita em paralelo. Nessa etapa, é utilizado um *script* desenvolvido por Saldanha [75] chamado ***genome2interval***, para gerar intervalos de tamanho fixo baseados no tamanho de cada cromossomo, os quais serão utilizados na fase seguinte.

Por fim, no quarto passo, são gerados histogramas do processamento dos arquivos de saída da segunda e terceira fase, que indicam o número de fragmentos mapeados por intervalo dos cromossomos com a ferramenta ***coverageBed***, da *suite* BEDTools [67].

Os 24 cromossomos do genoma de referência humano (**hg19**) foram obtidos do banco de dados NCBI (*National Center for Biotechnology Information*) [20], e a descrição do genoma pode ser obtida em `http://www.ncbi.nlm.nih.gov/genome/assembly/293148/`.

6.1.1 Execuções Realizadas

Para realizar os testes nos ambientes já citados na seção anterior, além da definição e da montagem do *workflow* apresentado na Figura 6.1, foram realizados as seguintes sequências, demonstradas nas Figuras 6.2, 6.3 e 6.4.

Após a definição e a montagem do SLA, o usuário pode optar por ele mesmo realizar as escolhas das máquinas que deseja executar seu *workflow*, para isso o sistema possui a tela apresentada na Figura 6.2, na qual optou-se, nesse exemplo , por escolher duas máquinas do primeiro ambiente, uma na Amazon e outra na Google.

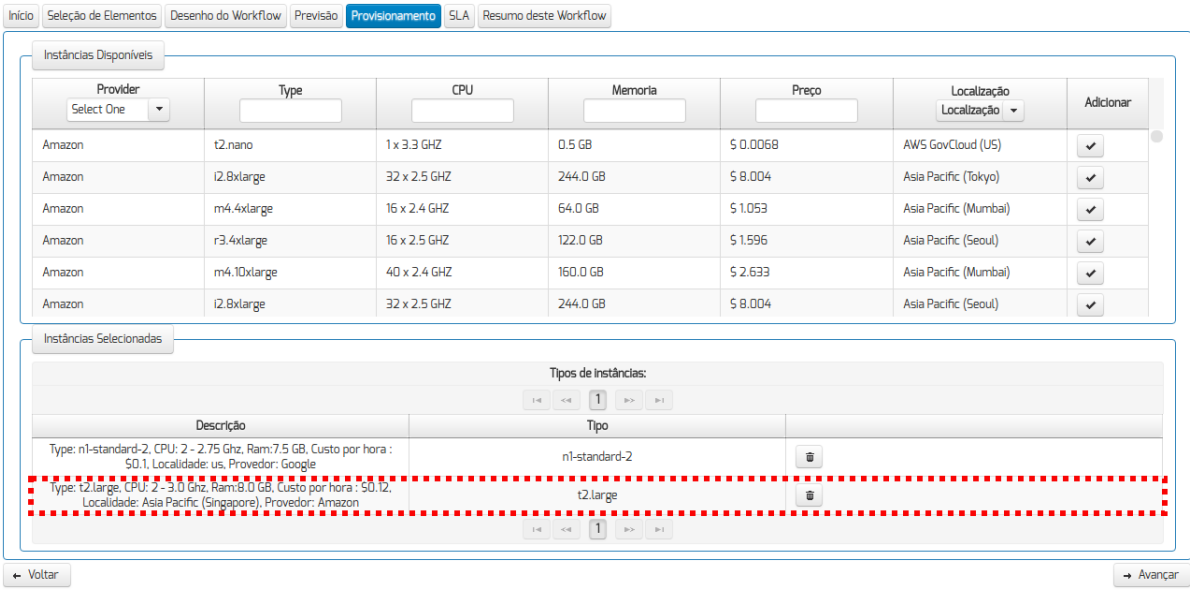

Figura 6.2: Escolha das Máquinas.

Assim, definido o ambiente e as configurações das máquinas, é apresentado o contrato para o cliente. Nessa tela o usuário tem a possibilidade de limitar a execução do *workflow* por tempo ou por custo. Além disso, caso o *workflow* não termine no período estipulado, o usuário tem a opção de pagar a mais para terminar a execução, ou remover as instâncias criadas pelo sistema.

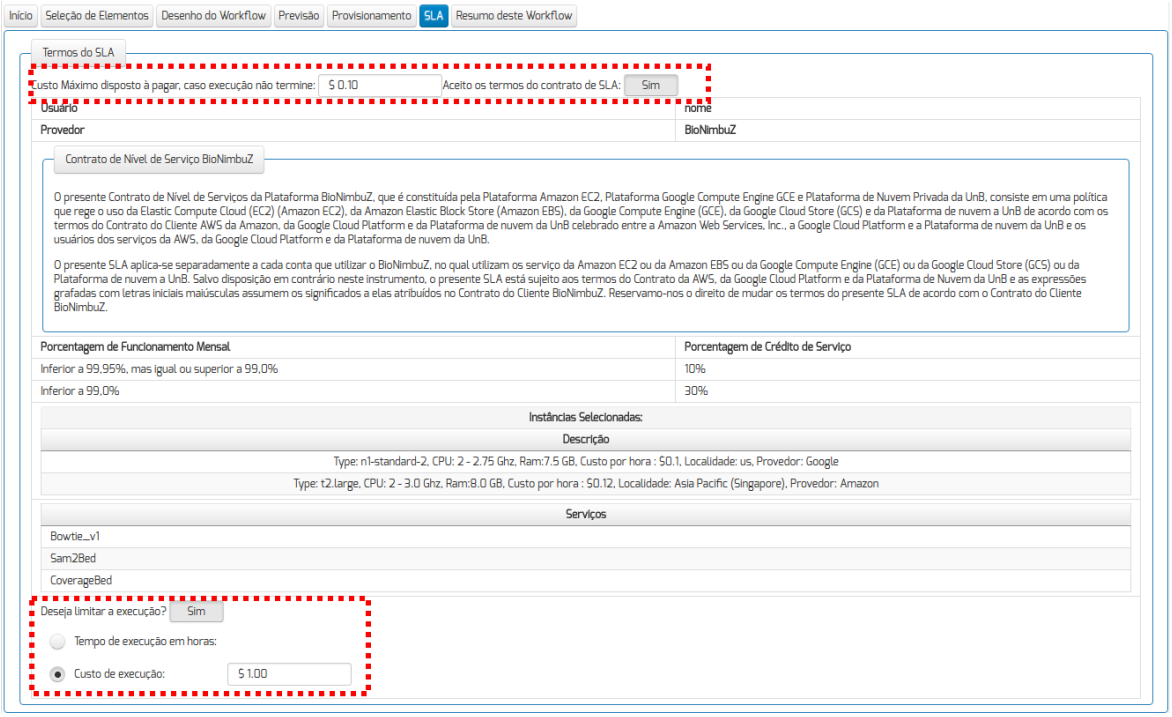

Figura 6.3: Contrato de SLA.

Com o contrato aceito e definido, é apresentado um resumo do *workflow* e um botão de confirmação para a execução do *workflow*, apresentado na Figura 6.4.

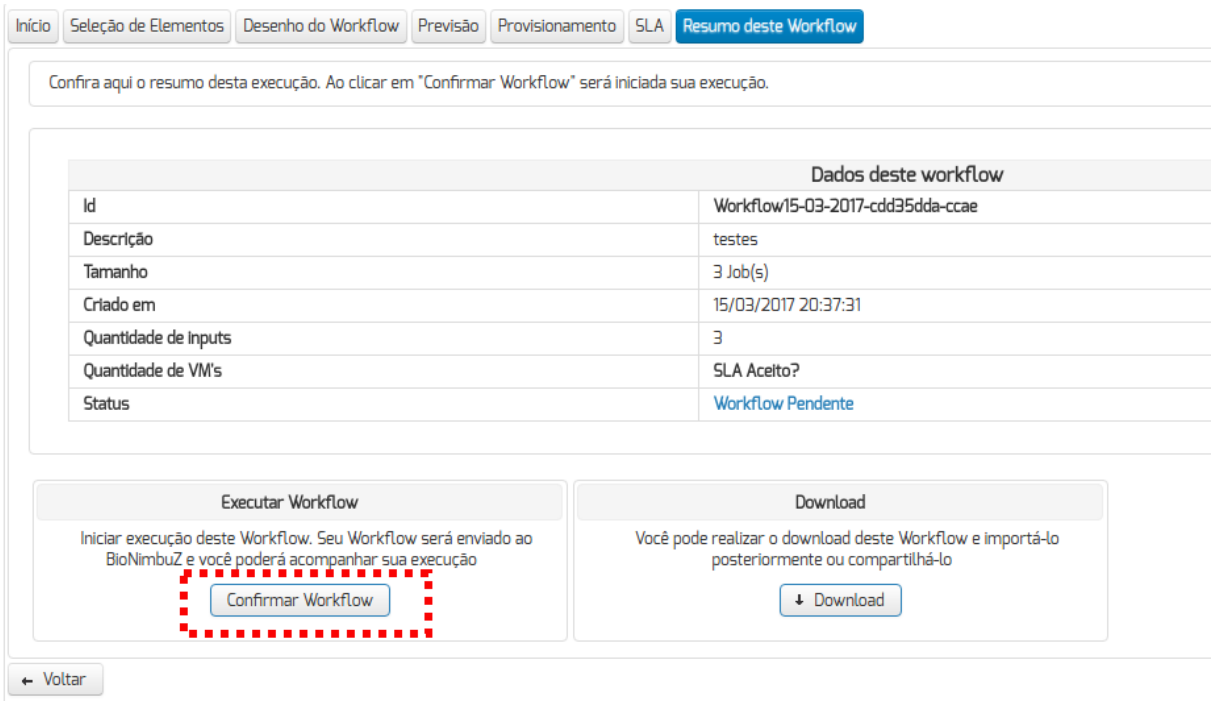

Figura 6.4: Confirmação do *Workflow*.

Com a confirmação da execução do *workflow*, é iniciado o processo de execução e monitoramento do SLA e da execução das tarefas, por meio de um gerenciamento de contrato, para que o mesmo seja respeitado. Todavia, caso ocorra quebra de contrato, são tomadas as medidas cabivéis. A Seção 6.2 apresenta o que ocorre caso ocorra essa quebra de contrato.

6.2 Quebra de Contrato - Relatório

Para gerenciar a quebra de contrato, por parte do sistema, o Controlador de SLA monitora constantemente os SLAs dos usuários. Assim, o controlador verifica se as tarefas foram executadas por completo, ou se a limitação imposta pelo usuário foi atingida antes da conclusão. Assim sendo, existem alguns relatórios que são informados para o usuário a respeito da máquina que contratou, e do tempo em que a tarefa está em execução.

Quando as configurações das máquinas contratadas não são as configurações ofertadas pelos provedores, é gerada uma mensagem para o usuário, no detalhe do *workflow*, para avisá-lo de que aquelas configurações não correspondem ao que foi contratado, como pode ser observado na Figura 6.5.

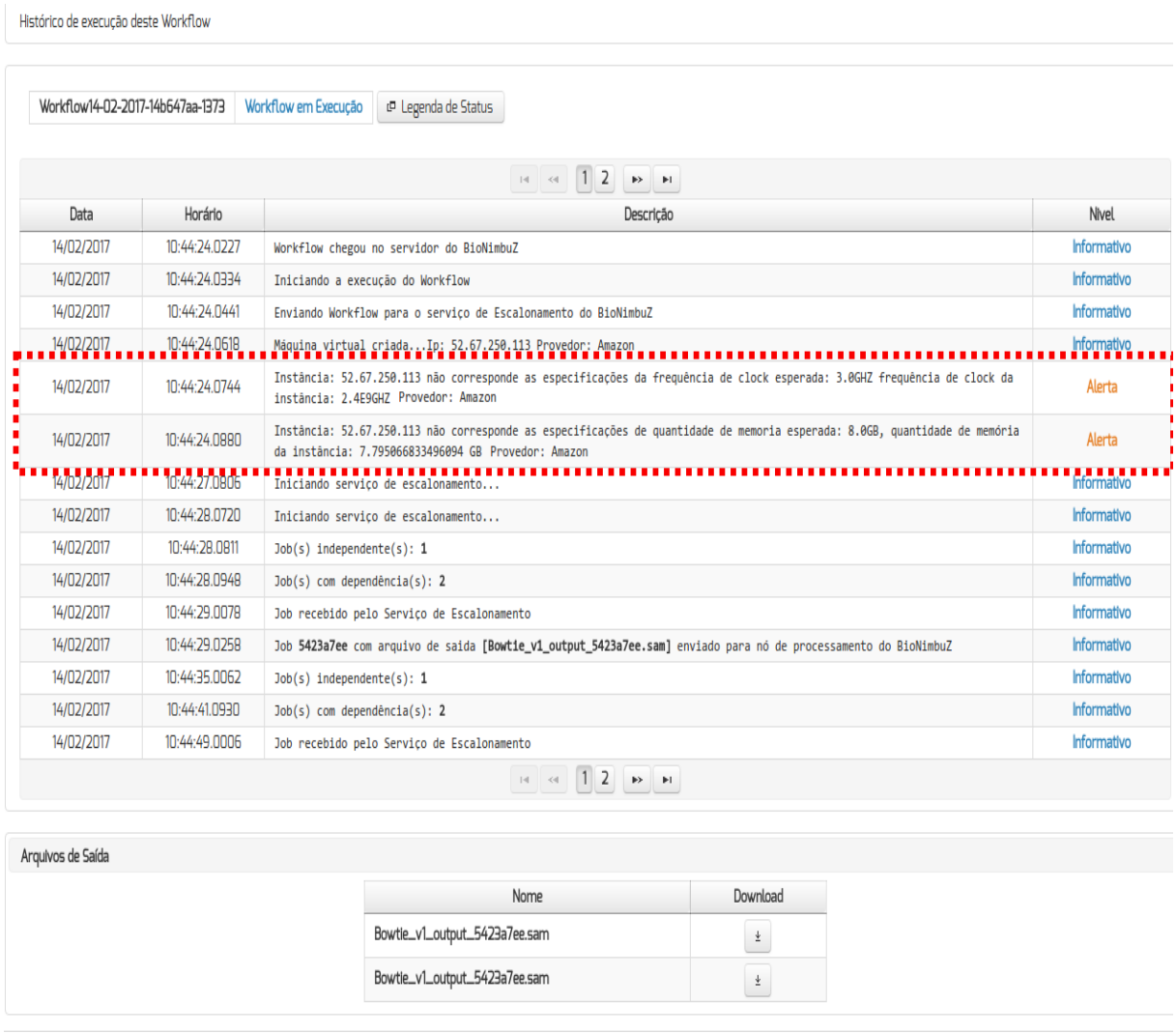

Figura 6.5: Configurações não Cumpridas pelo Provedor de Nuvem.

Essas informações detalham as configurações que não foram atendidas pela oferta. Informando, assim, ao usuário que a máquina instanciada pelo provedor não possui exatamente as configurações que foram ofertadas, conforme a parte destacada na Figura 6.5 demonstra. Assim, a plataforma de federação pode entrar em contato com o provedor de nuvem para tomar as medidas cabivéis.

Além disso, quando o custo total do *workflow* for atingido, vão ser disparadas informações sobre essa situação, e o *workflow* poderá ser interrompido. Dessa forma, as instâncias definidas para essa execução são removidas, como pode ser observado na Figura 6.6.

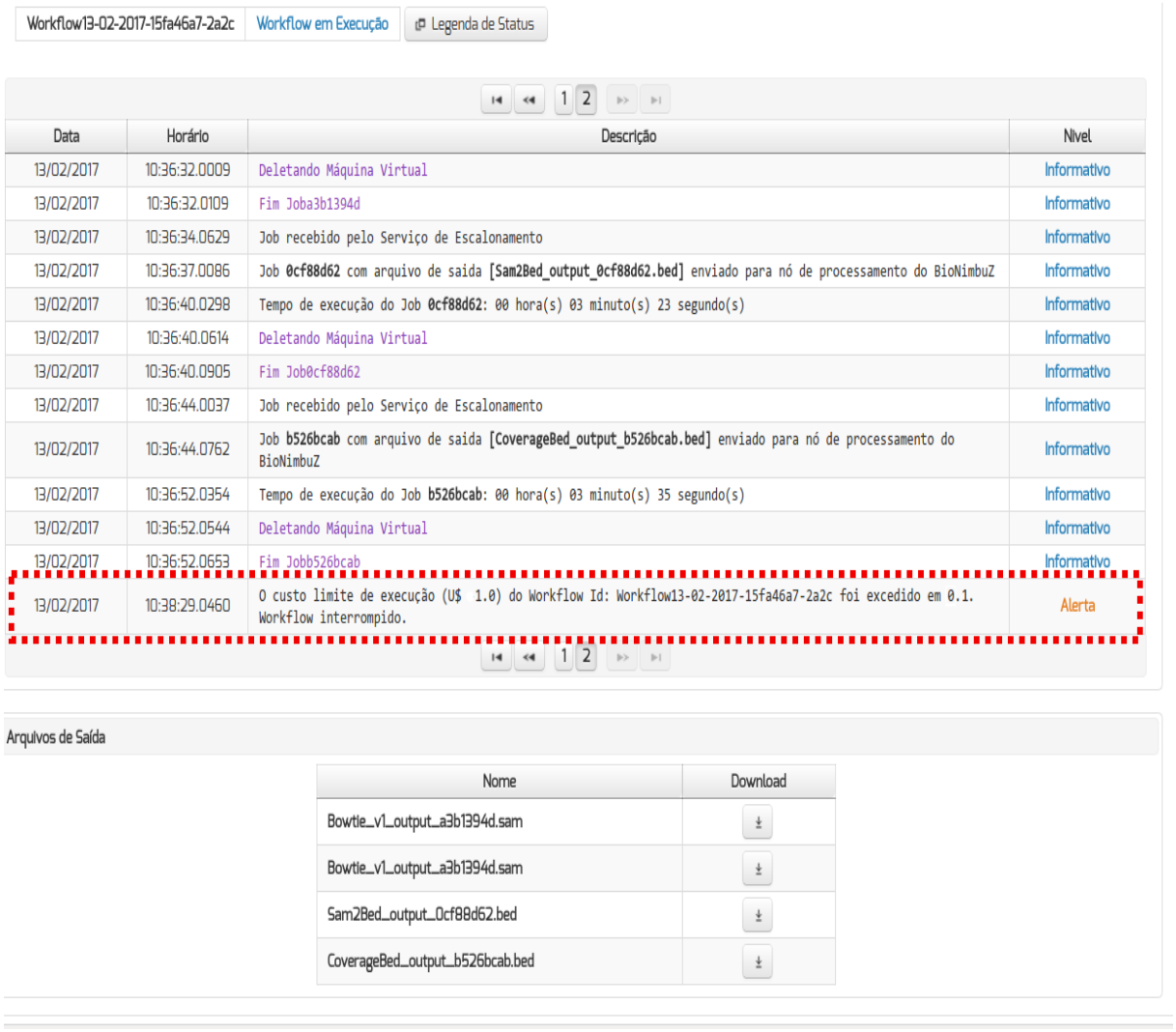

Figura 6.6: Limite de Custo do *Workflow* Excedido.

Essas mensagens possuem as informações de quanto era o limite do custo imposto, qual *workflow* ultrapassou esse limite, e de quanto foi excedido o limite. Assim, conforme observado em destaque na Figura 6.6, um *workflow* com limite de $U\$$ 1.0 foi ultrapassado em $U\$$ 0.1, fazendo com que o *workflow* seja interrompido e suas instâncias removidas.

Como resultado da execução do *workflow*, foi testado também o *overhead* que a adição do controlador na plataforma provoca, o qual é demonstrado na Seção 6.3.

6.3 *Overhead* Adicionado à Plataforma

Os testes realizados nesta seção objetivam analisar o *overhead* causado pela inserção do Controlador de SLA na plataforma BioNimbuZ. Para os testes, cada um dos ambientes

montados executou dois cenários. No primeiro cenário foi executado o *workflow* sem o controlador de SLA, obtendo, assim, a média do tempo de execução do *workflow*, que é demonstrada na Figura 6.7. No segundo cenário foi realizada a execução do mesmo *workflow*, esse com o Controlador de SLA, para obter o tempo de *overhead*, adicionado a execução do *workflow*.

Com os resultados apresentados na Figura 6.7, pode-se concluir que a adição do Controlador de SLA, não gerou nenhum *overhead* significativo na execução do *workflow*. Isso se deve ao fato de que o controlador não interfere na execução do *workflow*, pois o monitoramento é feito pelas informações mantidas pelo ZooKeeper [25], preservando, assim, o tempo de execução das tarefas, e monitorando o contrato de forma transparente.

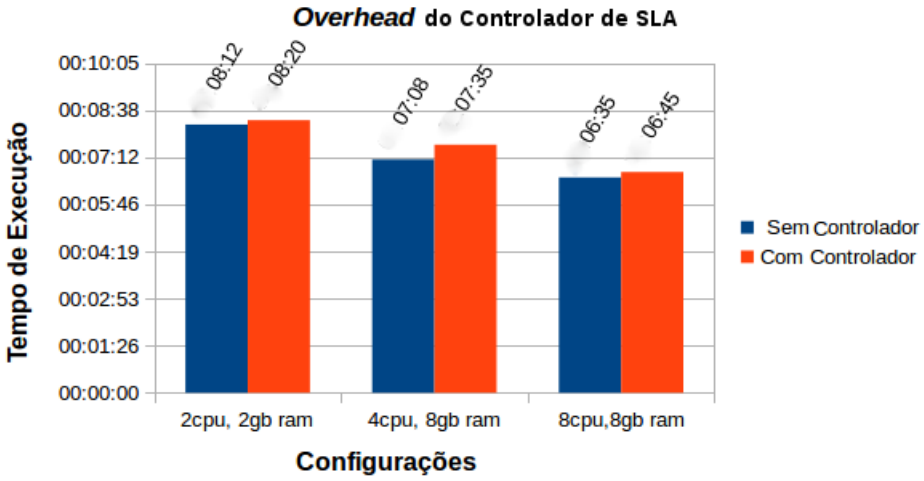

Figura 6.7: *Overhead* Adicionado à Plataforma BioNimbuZ.

Além disso, na Figura 6.7 pode ser observado que o maior percentual de *overhead* obtido foi no ambiente composto por duas máquinas com 4 CPUs, 8 GB de RAM, que gerou um *overhead* de aproximadamente 6% em relação ao tempo de execução sem o Controlador de SLA.

6.4 Custo das Instâncias

Como apresentado no Capítulo 5, o Controlador de SLA possui um serviço que busca uma lista de todas as instâncias ofertadas pelos provedores e disponibiliza essa lista por meio de uma interface *REST*. Para realizar os testes, o controlador foi integrado a plataforma BioNimbuZ. Portanto, a plataforma consome e apresenta essas informações de forma mais amigável, e isso gerou a possibilidade do usuário escolher mais adequadamente a máquina a ser ativada para a sua aplicação.

Para mostrar esse auxílio ao usuário, na Figura 6.8 é possivel notar que o usuário pode filtrar as configurações desejadas. Assim, ao invés do usuário ter que acessar cada provedor para comparar os preços das instâncias, o controlador faz a busca das ofertas em cada provedor da federação, apresentando então, essas ofertas em forma de lista, e permitindo que o usuário realize uma filtragem. Essa filtragem será feita de acordo com os critérios definidos por cada usuário.

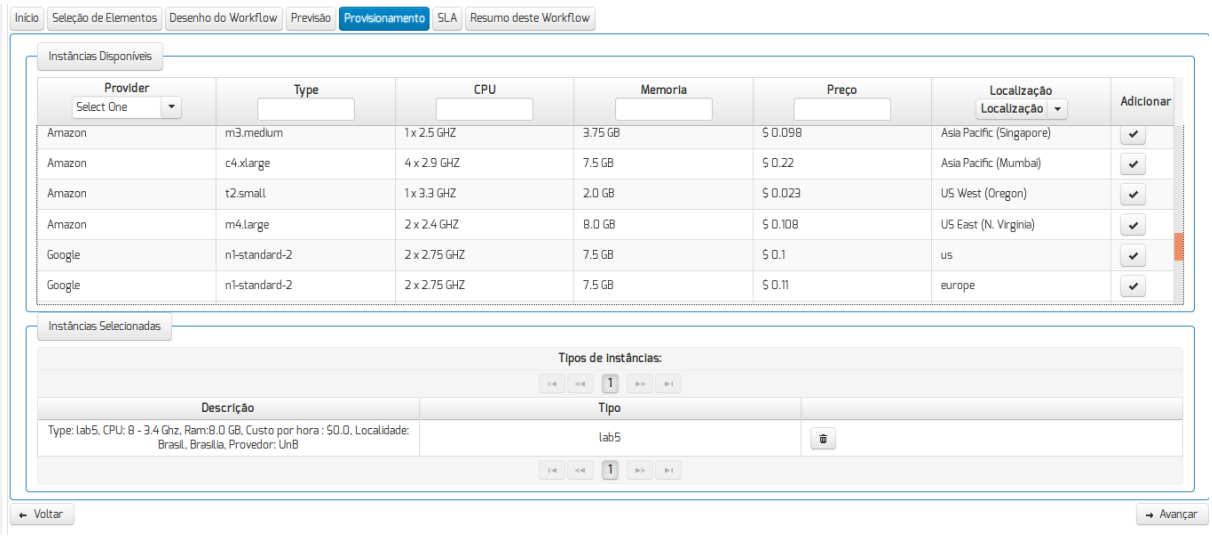

Figura 6.8: Exemplo de uma Lista com as Instâncias Fornecidas pela Controlador de SLA, com seus Filtros de Configurações.

E para demonstrar a diferença entre as instâncias e seus preços, um conjunto de instâncias foram selecionadas e apresentadas na Tabela 6.1. Com essa relação é possível notar que há uma variação considerável nos preços das máquinas, considerando os critérios provedor, tipo da instância, quantidade de núcleos, frequência desses núcleos, quantidade de memória RAM, custo por hora da instância em dólar e localidade. Essa lista foi gerada a partir do serviço de busca implementado pelo controlador de SLA, e ela pode variar de acordo com os requisitos de cada usuário.

Tabela 6.1: Tabela Filtrada de Instâncias.

Provedor	Instância	Núcleos	Clock	RAM	Custo (U$)	Localidade
Google	n1-highmem-8-preemptible	2	2.75	7.5	0.1	USA
Google	n1-highmem-8-preemptible	2	2.75	7.5	0.11	Europa
Google	n1-standard-1	2	2.75	7.5	0.11	Ásia
Amazon	t2.large	2	3.0	8	0.094	USA
Amazon	m4.xlarge	2	3.0	8	0.101	Europa

Amazon	n1-highmem-16	2	2.4	8	0.132	Ásia
Google	n1-standard-16-preemptible	4	2.75	15	0.2	USA
Google	n1-standard-16-preemptible	4	2.75	15	0.22	Europa
Google	n1-standard-2	4	2.75	15	0.22	Ásia
Amazon	c4.xlarge	4	2.4	16	0.215	USA
Amazon	n1-highmem-2-preemptible	4	2.4	16	0.238	Europa
Amazon	c3.xlarge	4	2.4	16	0.265	Ásia
Google	n1-standard-16	8	2.75	30	0.4	USA
Google	n1-standard-16	8	2.75	30	0.44	Europa
Google	n1-standard-16-preemptible	8	2.75	30	0.44	Ásia
Amazon	m3.xlarge	8	2.4	32	0.431	USA
Amazon	i2.xlarge	8	2.4	32	0.475	Europa
Amazon	n1-highcpu-16-preemptible	8	2.4	32	0.528	Ásia

No caso das instâncias apresentas na Tabela 6.1, elas foram selecionadas pelo menor preço das localidades da Ásia, da Europa e dos Estados Unidos de cada provedor, e com as configurações: máquina com 2 CPU e 7,5 GB de memória RAM; máquina com 4 CPU e 15 GB de memória RAM; e máquina com 8 CPU e 30 GB de memória RAM. Analisando, então, as diferenças de preço entre essas instâncias. A Figura 6.9 mostra a variação de custo entre regiões e provedores de nuvem.

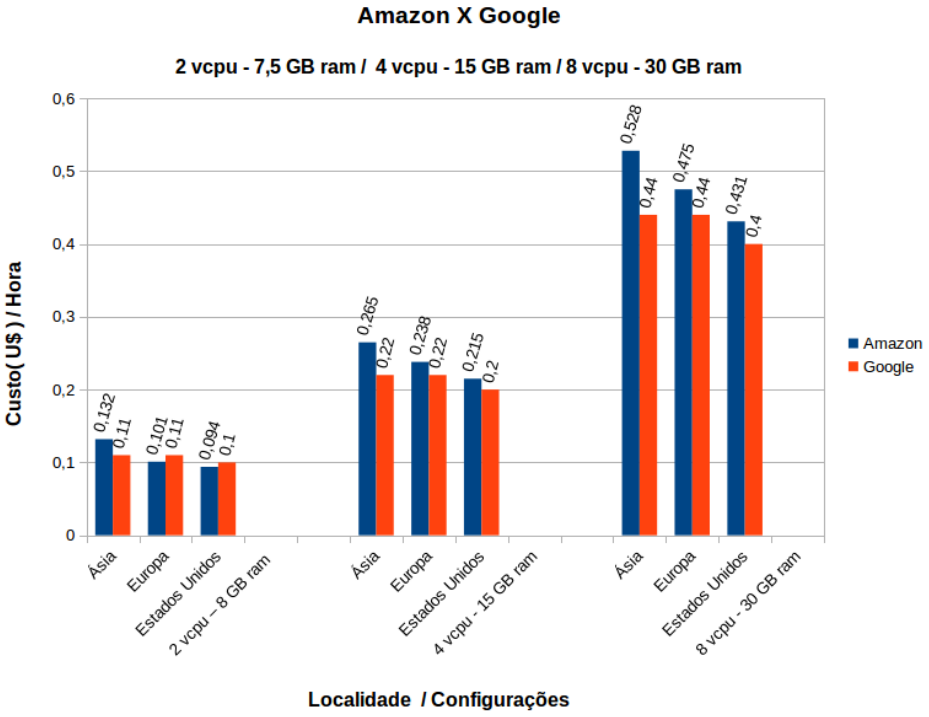

Figura 6.9: Preços das Instâncias da Amazon X Google.

Diante dos dados apresentados, a Figura 6.10 mostra o percentual na variação de preço entre essas máquinas, considerando o provedor e a localidade

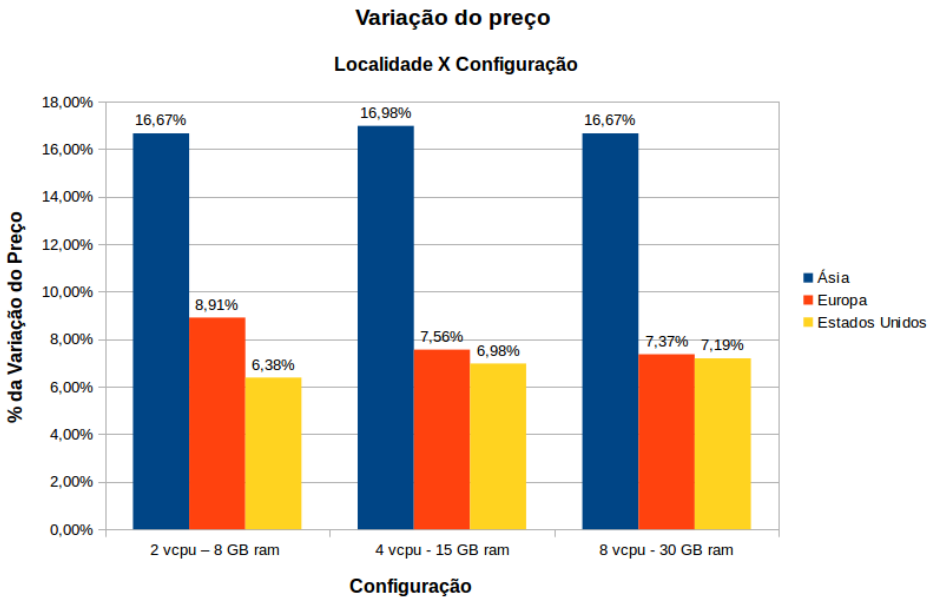

Figura 6.10: Variação de Preço entre Amazon e Google, por Localidades e Configurações.

Como pode ser observado na Figura 6.10, há uma variação de preço entre os provedores que pode chegar a 17%, aproximadamente. Isso faz com que os usuários tenham a facilidade e a liberdade de escolher uma configuração de máquina que atenda suas necessidades e, assim, contratar recursos com preços menores.

Dessa forma, observar-se que o controlador de SLA auxilia na facilidade de escolha das máquinas pelo usuário, fazendo o papel de *broker*. Pois assim, o usuário pode filtrar melhor suas configurações, e escolher a máquina que atende suas necessidades com a possibilidade de contratar as mesmas configurações com preços menores.

6.5 Considerações Finais

Neste capítulo foram apresentados os testes realizados para avaliar os resultados da implementação do controlador, utilizando como estudo de caso a plataforma BioNimbuZ.

Além de avaliar qual o impacto da adição do controlador a uma plataforma de nuvens federadas, também foi possível observar nos testes o auxílio que o controlador trouxe para o usuário ao disponibilizar todas as instâncias ofertadas pelos provedores, permitindo assim uma busca rápida em todas as ofertas para que o usuário possa escolher a melhor oferta que atende suas necessidades.

Capítulo 7

Conclusão e Trabalhos Futuros

Neste trabalho foi proposto um controlador de SLA para plataforma de nuvens federadas. O controlador proposto foi implementado baseado no ciclo de seis etapas, citado em Buyya *et al.* [90], que utiliza uma sequência de etapas essenciais para o funcionamento de um SLA.

A implementação de um Controlador de Acordos de Níveis de Serviço atua, principalmente, na satisfação dos clientes e na garantia da Qualidade de Serviço a ser entregue pela plataforma de nuvens federadas. Para isto, o controlador proposto neste trabalho levou em consideração os parâmetros de qualidade de serviço que o usuário necessita, tais como o quanto o usuário está disposto a pagar, por quanto tempo deseja que as instâncias fiquem ativas para executar a aplicação, a quantidade de máquinas e as configurações das máquinas que o usuário deseja que execute sua aplicação.

Assim, o controlador proposto garante o cumprimento dos acordos de serviço realizados em cada execução de todos os usuários. É importante ressaltar que o *overhead* inserido na execução das aplicações pelo controlador de SLA foi insignificante. Isso garante que não haverá influência no aumento do custo final pelo uso do controlador.

Além disso, pode-se concluir que a implementação do controlador de SLA facilitou as escolhas das máquinas virtuais para as execuções de tarefas, pois um dos serviços do controlador é disponibilizar a lista de instâncias ofertadas pelos provedores de nuvem, permitindo, assim, que o usuário filtre as configurações de máquina que ele deseja contratar. Essa filtragem é realizada por meio de buscas disponíveis na apresentação das listas de instâncias.

Como os preços das instâncias ofertadas pelos provedores podem variar aproximadamente em até 17%, para a mesma instância em diferentes provedores, os usuários tem a possibilidade de escolher menores preços para as configurações desejadas por ele, implicando, consequentemente, em uma redução no custo da execução das tarefas. Além de garantir que as máquinas contratadas para aquela execução tenham as configurações mantidas pelos provedores, disponibilizando assim informações mais detalhadas dessas configurações contratadas para o usuário.

Como trabalho futuro propõe-se a inserção de novos parâmetros de qualidade de serviço, como por exemplo a segurança no tráfego dos dados. Além disso, pretende-se implantar um banco de dados distribuídos para melhorar o acesso das informações geradas pelo serviço de

predição. Outro ponto importante a ser melhorado é a utilização de um serviço que faça *Single Sign-on* [62] com as contas dos usuários em provedores de nuvem, facilitando assim, a gerência da fatura.

Referências

[1] Altintas, I., Barney, O., and Jaeger-Frank, E. Provenance collection support in the kepler scientific workflow system. In *Proceedings of the 2006 International Conference on Provenance and Annotation of Data* (Berlin, Heidelberg, 2006), IPAW'06, Springer-Verlag, pp. 118–132. 39

[2] Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. Above the clouds: A Berkeley view of cloud computing. Tech. rep., Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 2009. 11

[3] Azevedo, D. R., and Freitas Júnior, T. B. Uma nova política de armazenamento para a plataforma BioNimbuZ de nuvem federada. `http://bdm.unb.br/handle/10483/13199`, 2015. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 2, 23, 24, 32

[4] Bahga, A., and Madisetti, V. *Cloud Computing: A Hands-On Approach.* CreateSpace Independent Publishing Platform, 2013. 14

[5] Bardhan, S., and Milojicic, D. A mechanism to measure quality-of-service in a federated cloud environment. In *Proceedings of the 2012 Workshop on Cloud Services, Federation, and the 8th Open Cirrus Summit* (New York, NY, USA, 2012), FederatedClouds '12, ACM, pp. 19–24. 2

[6] Barreiros Júnior, W. O. Escalonador de tarefas para o plataforma de nuvens federadas BioNimbuZ usando beam search iterativo multiobjetivo. `http://bdm.unb.br/handle/10483/13146`, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 2, 23, 24, 32

[7] Berman, F., Fox, G., and Hey, A. J. G. *Grid Computing: Making the Global Infrastructure a Reality.* John Wiley & Sons, Inc., New York, NY, USA, 2003. 9, 15

[8] Buco, M. J., Chang, R. N., Luan, L. Z., Ward, C., Wolf, J. L., and Yu, P. S. Utility computing SLA management based upon business objectives. *IBM Systems Journal 43*, 1 (2004), 159–178. 1, 41

[9] Buyya, R. *High Performance Cluster Computing: Programming and Applications*, 1st ed., vol. 2. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999. x, 7, 8

[10] Buyya, R., Ranjan, R., and Calheiros, R. N. 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 - Volume Part I* (Berlin, Heidelberg, 2010), ICA3PP'10, Springer-Verlag, pp. 13–31. x, 1, 2, 18, 19, 21, 22, 46, 47, 48, 50

[11] Buyya, R., Yeo, C. S., Venugopal, S., Broberg, J., and Brandic, I. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst. 25*, 6 (jun. 2009), 599–616. xii, 9, 11, 16, 17

[12] Celesti, A., Tusa, F., Villari, M., and Puliafito, A. How to enhance cloud architectures to enable cross-federation. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on* (jul. 2010), pp. 337 –345. x, 18, 19, 20, 21

[13] Celesti, A., Tusa, F., Villari, M., and Puliafito, A. Three-phase cross-cloud federation model: The cloud sso authentication. In *2010 Second International Conference on Advances in Future Internet* (July 2010), pp. 94–101. 18

[14] Coulouris, G., Dollimore, J., and Kindberg, T. *Sistemas distribuidos.* Addison Wesley, 2001. 6

[15] Crockford, D. Json. `http://json.org/`. Acessado online em 05 de junho de 2016. 36, 52, 65

[16] Factor, M., Meth, K., Naor, D., Rodeh, O., and Satran, J. Object storage: the future building block for storage systems. In *2005 IEEE International Symposium on Mass Storage Systems and Technology* (June 2005), pp. 119–123. 33

[17] Falasi, A. A., Serhani, M. A., and Dssouli, R. A model for multi-levels SLA monitoring in federated cloud environment. In *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)* (Dec 2013), pp. 363–370. 42, 48, 50

[18] Feo, T. A., and Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6*, 2 (1995), 109–133. 36

[19] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. Hypertext Transfer Protocol–http/1.1, Request For Coments (RFC 2616). `http://tools.ietf.org/pdf/rfc2616.pdf`, 1999. Acessado online em 05 de junho de 2016. 36

[20] for Biotechnology Information, N. C. NCBI. `http://www.ncbi.nlm.nih.gov`. Acessado online em 17 de janeiro de 2017. 69

[21] Foster, I., Kesselman, C., and Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications 15*, 3 (2001), 200–222. x, 10

[22] Foster, I., Zhao, Y., Raicu, I., and Lu, S. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08* (nov. 2008), pp. 1 –10. x, 10, 11, 13, 15, 16

[23] Foundation, A. S. Apache avro). `http://avro.apache.org/`. Acessado online em 05 de junho de 2016. 24, 36

[24] Foundation, N. S. Open science grid. `http://www.opensciencegrid.org/`, 2013. Acessado online em 27 de abril de 2016. 9

[25] Foundation, T. A. S. Apache zookeeper. `http://zookeeper.apache.org/`, 2010. Acessado online em 05 de setembro de 2016. 24, 31, 32, 36, 37, 74

[26] Fundation, T. A. S. Apache avro documentation. http://avro.apache.org/docs/1.8.1/. Acessado online em 05 de junho de 2016. 36, 37

[27] Fundation, T. A. S. Apache fundation. http://apache.org/. Acessado online em 30 de março de 2016. 36

[28] Gallon, R. F. Política de armazenamento de dados em nuvens federadas para dados biológicos. Master's thesis, School, 2014. 32

[29] Google. Cloud storage. https://cloud.google.com/storage/. Acessado online em 10 de janeiro de 2017. 33

[30] Google. Google app engine. https://developers.google.com/appengine/docs/whatisgoogleappengine?hl=pt-br. Acessado online em 30 de março de 2016. 14

[31] Google. Google cloud plataform. https://cloud.google.com/. Acessado online em 10 de janeiro de 2017. 18, 38

[32] Google. Google docs. https://docs.google.com/?hl=pt-BR. Acessado online em 27 de abril de 2016. 14

[33] Google. Google compute engine. https://cloud.google.com/compute/, jan 2017. Acessado online em 10 de janeiro de 2017. 38, 64, 65

[34] Hollingsworth, D., Services, F., and Kingdom, U. The workflow reference model: 10 years on. In *Fujitsu Services, UK; Technical Committee Chair of WfMC* (2004), pp. 295–312. 39, 40

[35] Illumina, I. Illumina. http://www.illumina.com. Acessado online em 17 de janeiro de 2017. 68

[36] JCraft. Pure implementation of sftp for java. http://www.jcraft.com/jsch/examples/Sftp.java.html, 2012. Acessado online em 10 de janeiro de 2017. 24

[37] Jin, L. J., and Machiraju, V. A. Analysis on service level agreement of web services. Tech. rep., HPL-2002-180, Software Technology Laboratories, HP Laboratories, 2002. 42

[38] Jrad, F., Tao, J., and Streit, A. SLA based Service Brokering in Intercloud Environments. In *CLOSER* (2012), F. Leymann, I. Ivanov, M. van Sinderen, and T. Shan, Eds., SciTePress, pp. 76–81. 41, 48, 50

[39] Junqueira, F., and Reed, B. *ZooKeeper: distributed process coordination.* "O'Reilly Media, Inc.", 2013. 37

[40] Kerteśz, A., Kecskemeti, G., and Brandic, I. Autonomic SLA-Aware Service Virtualization for Distributed Systems. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on* (Feb 2011), pp. 503–510. 48

[41] Kwei-Jay, L., and Gannon, J. Atomic Remote Procedure Call. *Software Engineering, IEEE Transactions on SE-11*, 10 (1985), 1126–1135. 24

[42] Lages, N. A. C., and Nogueira, J. M. S. *Introdução aos sistemas distribuídos.* Ed. da Unicamp/Papirus, 1986. 6, 15

[43] Langmead, B., Trapnell, C., Pop, M., and Salzberg, S. Ultrafast and memory-eficient alignment of short dna sequences to the human genome. http://bowtie-bio.sourceforge.net/index.shtml, 2009. Acessado online em 10 de janeiro de 2017. 69

[44] Lenk, A., Klems, M., Nimis, J., Tai, S., and Sandholm, T. 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* (2009), IEEE Computer Society, pp. 23–31. 12

[45] LLC, A. W. S. Amazon Elastic Compute Cloud (EC2). http://aws.amazon.com/pt/ec2/. Acessado online em 10 de agosto de 2016. 14, 18, 38

[46] LLC, A. W. S. Amazon Simple Storage Service (Amazon S3). http://aws.amazon.com/s3/. Acessado online em 29 de maio de 2016. 14, 33

[47] LLC, Z. Zabbix. http://www.zabbix.com/rn2.2.0, 2013. Acessado online em 17 de janeiro de 2017. 49

[48] Mann, C. The end of Moore's law. *Technology Review 103*, 3 (2000), 42–48. 5

[49] Marioni, J. C., M., C. E., Mane, S. M., Stephens, M., and Gilad, Y. RNA-Seq: An assessment of technical reproducibility and comparison with gene expression arrays. *Genome Research 18* (2008), 1509–1517. 68

[50] Mell, P., and Grance, T. The NIST definition of cloud computing. *National Institute of Standards and Technology 53*, 6 (2009), 50. 11

[51] Mell, P., and Grance, T. The nist definition of cloud computing. Tech. Rep. 800-145, National Institute of Standards and Technology (NIST), Gaithersburg, MD, September 2011. 8

[52] Metsch, T., Edmonds, A., and Bayon, V. Using Cloud Standards for Interoperability of Cloud Frameworks. Tech. rep., SLA@SOI, 2010. Acessado online em 01 de janeiro de 2017. 48

[53] Microsoft. Windows azure. http://www.windowsazure.com/pt-br/. Acessado online em 05 de junho de 2016. 18

[54] Moore, G. E. Lithography and the future of moore's law. *IEEE Solid-State Circuits Society Newsletter 20*, 3 (Sept 2006), 37–42. 5

[55] Morley, M. Json-rpc. http://json-rpc.org/wiki/about, 2004. Acessado online em 17 de janeiro de 2017. 49

[56] Moura, B., Lima, D., Ribeiro, E., Aráujo, A. P. F., Walter, M. E., Holanda, M. T., and Oliveira, G. A Storage Policy for a Hybrid Federated Cloud Platform executing Bioinformatics Applications. *C4BIE 2014: Cloud for Business, Industry and Enterprises* (May 2014). Proceedings of the 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2014), Chicago, USA. 1, 2, 23, 24

[57] Moura, B. R., and Bacelar, D. L. Política para armazenamento de arquivos no ZooNimbus. http://bdm.unb.br/handle/10483/6469, 2013. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 32

[58] Moustafa, S., Elgazzar, K., Martin, P., and Elsayed, M. Slam: Sla monitoring framework for federated cloud services. In *2015 IEEE/ACM 8th International Conference on Utility and Cloud Computing (UCC)* (Dec 2015), pp. 506–511. 49, 50

[59] Neph, S., and Reynolds, A. Sam2Bed. `http://bedops.readthedocs.io/en/latest/content/reference/file-management/conversion/sam2bed.html`. Acessado online em 17 de janeiro de 2017. 69

[60] Nowicki, B. NFS: Network File System Protocol Specification, Request For Coments (RFC 1094). `http://tools.ietf.org/pdf/rfc1094.pdf`, 1989. Acessado online em 29 de maio de 2016. 16

[61] Oliveira, G. S. S. Acosched: um escalonador para o ambiente de nuvem federada Zoonimbus. `http://bdm.unb.br/handle/10483/6477`, 2013. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 32

[62] Pashalidis, A., and Mitchell, C. J. A taxonomy of single sign-on systems. In *Australasian Conference on Information Security and Privacy* (2003), Springer, pp. 249–264. 79

[63] Paula, R. Proveniência de dados em workflows de bioinformática. Master's thesis, Universidade de Brasília - UnB, 2013. x, 40

[64] Pitanga, M. *Construindo supercomputadores com Linux*. Brasport, 2004. 6

[65] Platform, G. C. Compute engine api client library for java. `https://developers.google.com/api-client-library/java/apis/compute/v1`, 2017. Acessado online em 17 de janeiro de 2017. 65

[66] prenkels, R., and Pras, A. Service level agreements : Internet ng deliverable d2.7. Report, Electrical Engineering, Mathematics and Computer Science (EEMCS), Enschede, the Netherlands, 2001. University of Twente, Centre for Telematics and Information Technology (CTIT). 44

[67] Quinlan, A. R. BedTools. `http://bedtools.readthedocs.org/en/latest/`. Acessado online em 17 de janeiro de 2017. 69

[68] Ramos, V. A. Um sistema gerenciador de workflows científicos para a plataforma de nuvens federadas BioNimbuZ. `http://bdm.unb.br/handle/10483/13145`, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. x, xi, 2, 23, 24, 26, 27, 28, 29, 30, 35, 38, 39, 66

[69] RedHat, J. Resteasy. `http://resteasy.jboss.org`. Acessado online em 07 de janeiro de 2017. 30, 63

[70] Rimal, B. P., Eunmi, C., and Lumb, I. A Taxonomy and Survey of Cloud Computing Systems. In *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on* (aug. 2009), pp. 44–51. 14

[71] Robinson, M. D., and Oshlack, A. A scaling normalization method for differential expression analysis of RNA-Seq data. *Genome Biol 11*, 3 (2010), R25. 68

[72] Rosa, M., Moura, B. R., Vergara, G., Santos, L., Ribeiro, E., Holanda, M., Walter, M. E., and Araújo, A. Bionimbuz: A federated cloud platform for bioinformatics applications. In *2016 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (Dec 2016), pp. 548–555. 2, 23

[73] Rosa, M. J. F. Predição aplicada em workflows científicos para estimativa de custos e recursos computacionais em nuvens federadas. Master's thesis, Universidade de Brasília - UnB, 2017. x, 35

[74] Saldanha, H., Ribeiro, E., Borges, C., Araújo, A., Gallon, R., Holanda, M., Walter, M. E., Togawa, R., and Setubal, J. C. *BioInformatics*. In Tech, 2012. 2, 22, 23

[75] Saldanha, H. V. Bionimbus: uma arquitetura de federação de nuvens computacionais híbrida para a execução de workflows de bioinformática. Master's thesis, Departamento de Ciência de Computação, Universidade de Brasília, 2012. 2, 18, 24, 69

[76] Santos, L. F. N. Novo serviço de armazenamento na plataforma de nuvem federada BioNimbuZ. `http://bdm.unb.br/handle/`, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 2, 23, 24, 33

[77] Sardenberg, R. S. Integridade e confidencialidade dos arquivos na plataforma de nuvem federada BioNimbuZ. `http://bdm.unb.br/handle/10483/13502`, 2016. Monografia de graduação, Departamento de Ciência de Computação, Universidade de Brasília. 35

[78] Services, A. W. Amazon web services. `http://aws.amazon.com/pt/`. Acessado online em 10 de janeiro de 2017. 38, 64, 65

[79] Services, A. W. Aws sdk para java. `https://aws.amazon.com/pt/sdk-for-java/`, 2013. Acessado online em 17 de janeiro de 2017. 65

[80] Sloan, J. D. *High performance Linux clusters with OSCAR, Rocks, openMosix, and MPI*. Rodopi, 2009. 5

[81] Sprenkels, R., Pras, A., Beijnum, B. J., and Goede, L. A customer service management architecture for the internet. Tech. Rep. TR-CTIT-00-17, Electrical Engineering, Mathematics and Computer Science (EEMCS), Enschede, the Netherlands, 2000. Acessado online em 10 de janeiro de 2017. 42

[82] Steen, M. V., and Tanenbaum, A. *Distributed Systems: Principles and Paradigms*. Prentice Hall; 2 edition, 2006. 6, 7, 24

[83] Sterling, T., Becker, D. J., Savarese, D., Dorband, J. E., Ranawake, U. A., and Packer, C. V. Beowulf: A parallel workstation for scientific computation. In *In Proceedings of the 24th International Conference on Parallel Processing* (1995), CRC Press, pp. 11–14. 7, 8

[84] Stoica, I., Morris, R., Liben-Nowell, D., Karger, D., Kaashoek, F., Dabek, F., and Balakrishnan, H. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on 11*, 1 (2003), 17–32. 23

[85] Van Der Aalst, W., and Van Hee, K. M. *Workflow management: models, methods, and systems*. MIT press, 2004. 39

[86] Vaquero, L. M., Merino, L. R., Caceres, J., and Lindner, M. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Comput. Commun. Rev. 39*, 1 (dec 2008), 50–55. x, 1, 11, 12

[87] Vergara, G. F. Arquitetura de um controlador de elasticidade para nuvens federadas. um estudo de caso na plataforma bionimbuz. Master's thesis, Universidade de Brasília - UnB, 2017. x, 31

[88] Verma, D. C. Service level agreements on IP networks. *Proceedings of the IEEE 92*, 9 (Sept 2004), 1382–1388. 42

[89] Wieder, P., Butler, J., Theilmann, W., and Yahyapour, R. *Service Level Agreements for Cloud Computing*. Springer New York, 2011. 48, 49, 50

[90] Wu, L., and Buyya, R. Service level Agreement (SLA) in utility computing systems. *Computing Research Repository - CoRR abs/1010.2881* (2010). x, 42, 43, 45, 47, 63, 78

[91] Youseff, L., Butrico, M., and Silva, D. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE'08* (2008), IEEE, pp. 1–10. 12

[92] Zhang, Q., Cheng, L., and Boutaba, R. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications 1*, 1 (Apr. 2010), 7–18. 8

[93] Zookeeper, A. Curator. `http://curator.apache.org/`. Acessado online em 10 de janeiro de 2017. 38