Política de Armazenamento em uma Infraestrutura de Nuvens Federadas para Aplicações de Bioinformática

Breno R. Moura, Deric L. Bacelar, Edward Ribeiro, Aletéia P. F. Araújo, Maristela T. Holanda and Maria E. M. T. Walter
Department of Computer Science University of Brasilia, UNB, Brazil
{mourabreno, dericlima, edward.ribeiro}@gmail.com, {aleteia, mholanda, mia}@cic.unb.br

Resumo*—Políticas de armazenamento são difíceis de serem implementadas para um ambiente de nuvens fede -radas, uma vez que existem muitos provedores componentes da federação com capacidades de armazenamento distintas que devem ser consideradas. Por outro lado, em Bioinformática, muitas ferramentas e bancos de dados necessitam de grandes volumes de recursos para processarem e armazenarem quantidades enormes de dados, que podem atingir facilmente** *terabytes* **de tamanho. Este trabalho trata do problema da política de armazenamento no BioNimbus, o qual é uma infraestrutura de nuvens federadas para aplicações de bioinformática. Neste contexto, este trabalho propõe uma política de armazenamento, chamada ZooClouS (ZooNimbus Cloud Storage***), que se baseia na latência, no custo, no** *uptime* **e no espaço livre de armazenamento para realizar uma escolha que distribui eficientemente os arquivos para os melhores recursos disponíveis na nuvem federada. Os experimentos foram realizados com dados biológicos reais, os quais foram executados em uma federação de nuvens constituídas com as nuvens da Amazon EC2, do Windows Azure e da Universidade de Brasília (UnB). Os resultados obtidos mostram que o ZooClouS conseguiu uma melhoria significativa no tempo de** *makespan* **das aplicações de Bioinformática executadas, quando comparado com a política de armazenamento aleatória que estava implementada no BioNimbus.**

I. Introdução

Em um cenário no qual a maior parte do processamento computacional é realizado por servidores locais e *data centers* proprietários, onde esse poder de processamento computacional nem sempre é utilizado por completo, paga-se por energia, manutenção e tecnologias que não são usadas em sua totalidade [4]. Neste contexto, surgiu a ideia de que, em vez de se pagar por uma estrutura proprietária, de manutenção cara, que as vezes fica ociosa, fosse utilizada uma estrutura que use os seus recursos de acordo com a necessidade de cada usuário. Assim, os recursos são alocados de acordo com a demanda de utilização necessária, a chamada computação sob demanda. A grande vantagem da computação sob demanda é que encerra o cenário onde o poder computacional disponível acaba sendo desperdiçado por não ser utilizado em sua totalidade [1].

Nesse cenário, surge a plataforma de computação em nuvem, a qual busca reduzir o custo computacional, aumentar a confiabilidade e a flexibilidade para transformar computadores em algo no qual todo o tratamento dos dados e aplicações são feitos como uma espécie de serviço básico [3]. O objetivo é que a computação em nuvem disponibilize um serviço totalmente sob demanda, em que para todo o processamento solicitado, os recursos sejam alocados recursos de acordo com a necessidade, e que esse processo seja totalmente dinâmico, oferecendo poder computacional somente na medida necessária.

Atualmente, é possível encontrar alguns serviços de nuvens públicas por meio de empresas como Amazon [14], Microsoft [5] e Google [10]. Em sistemas de computação em nuvem, diferentes fundamentos estão presentes, tais como virtualização, escalabilidade, intero -perabilidade, qualidade de serviço (QoS), mecanismo de tolerância a falhas e modelos de entrega de nuvens (privadas, públicas e híbridas). A estrutura de uma nuvem inclui diferentes partes envolvidas, com atributos e tecnologias que são acopladas para atender suas necessidades e seus diferentes tipos de serviços [17]. Os três principais tipos de serviços de nuvens disponibilizados são os de software (SaaS), infraestrutura (IaaS) e plataforma (PaaS) como serviços.

Como a quantidade de dados a ser processada aumenta cada vez mais no mundo da computação, seja em processos de aplicações ou no próprio armazenamento de dados, surge a necessidade de se ter um poder de processamento e armazenamento cada vez maior, que possa atender pedidos em um tempo hábil, de modo automático e dinâmico. Porém, uma nuvem possui recursos limitados e por conta disso surgiu a ideia de se integrar nuvens com o objetivo de aumentar os recursos disponíveis, pois se um recurso de uma nuvem se esgotar, uma outra nuvem pode utilizar os seus recursos caso estejam disponíveis. Dessa ideia emergiu o conceito de federação de nuvens.

As federações de nuvens 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 recurso para determinado processamento, uma outra nuvem que esteja com recursos disponíveis no momento pode receber este processamento.

Assim sendo, a federação de nuvens tem se mostrado uma técnica capaz de integrar diversas infraestruturas de nuvens, a qual proporciona a ilusão de que os recursos da nuvem são ilimitados.

Nesse cenário, Saldanha et al. [2] propôs uma arquitetura de federação de nuvens chamada BioNimbus. O BioNimbus é uma arquitetura simples e dinâmica que pode executar diferentes aplicações dentro da federação, atualmente ela tem sido utilizada para a execução de *workflows* de bioinformática.

O BioNimbus sofreu várias modificações na sua estrutura original, porém essas alterações não mudaram os objetivos e as propostas iniciais da arquitetura de federação em nuvens. Entre os objetivos do BioNimbus estão integrar e controlar diferentes provedores de infraestrutura e oferecer um serviço flexível e tolerante a falhas, com suas ferramentas de bioinformática. Todavia, a fim de modernizar o código do BioNimbus, foi integrado a arquitetura original o Apache ZooKeeper [9] e o Apache Avro [7]. Com essas mudanças o BioNimbus passou a ser chamado ZooNimbus. É importante ressaltar que o ZooNimbus, apesar das mudanças, continua a prover uma plataforma que constrói uma federação em nuvem e que executa serviços de bioinformática com recursos heterogêneos, públicos ou privados.

Assim, o objetivo deste trabalho é propor uma política de armazenamento de dados para a plataforma de fede - ração ZooNimbus. Para isso, este trabalho está dividido em seis Seções. A Seção II aborda as mudanças que ocorreram no ZooNimbus com a integração do Apache ZooKeeper. A Seção III apresenta as alterações que ocorreram na comunicação do ZooNimbus com a integração do Avro Apache. A Seção IV destaca a nova arquitetura do ZooNimbus. Em seguida, a Seção V explica como funciona a política de armazenamento de dados proposta neste artigo. Na Seção VI estão os resultados obtidos com os testes feitos com a implementação da política proposta. E por último, na Seção VII, estão algumas conclusões e os trabalhos futuros que foram propostos para a melhoria da política e do desempenho do ZooNimbus.

II. Apache ZooKeeper

ZooKeeper [9] é um serviço de coordenação de processos distribuídos, de código-aberto para aplicações distribuídas. ZooKeeper expõe um conjunto simples de primitivas que podem construir aplicações distribuídas com a implementação de serviços de nível superior como sincronização, manutenção, configuração e grupos de processos. Serviços de coordenação distribuídos são notoriamente difíceis de serem implementados. Além disso, estes serviços são propensos a erros, tais como condições de corrida e impasses. Assim, mesmo quando feito corretamente, diferentes implementações de serviços de coordenação levam a complexidade de gestão, quando os aplicativos são implantados. Neste contexto, o ZooKeeper visa facilitar às aplicações distribuídas a responsabilidade de implementar serviços de coordenação a partir do zero. O ZooKeeper foi projetado para armazenar dados de coordenação, informações de *status*, configurações, informações de localização, etc [9].

Com relação aos seus dados, o ZooKeeper utiliza o termo *znode*, que são usados para designar nós de dados no ZooKeeper. Os *znodes* são estruturas de dados semelhantes a diretórios. Os dados armazenados em cada *znode* são lidos e escritos atomicamente. Operações de leitura obtêm todos os *bytes* de dados associados a um *znode* e uma operação de gravação substitui todos os dados de forma atômica. Cada nó tem uma lista de controle de acesso (ACL), que restringe quem pode ler/escrever em cada *znode*. O ZooKeeper tem também a noção de *znodes* efêmeros. Estes *znodes* existem enquanto a sessão que criou o *znode* está ativa. Quando a sessão termina o *znode* é excluído, pois cada sessão correponde a um cliente conectado.

Outro conceito importante do ZooKeeper é o suporte a *watchers*, que são avisos de eventos. Os servidores podem definir um *watcher* em um *znode*. Um *watcher* será acionado e removido quando ocorrerem mudanças no *znode*. Quando um *watcher* é disparado, o cliente recebe um pacote dizendo que o *znode* mudou. E se a conexão entre o cliente e um dos servidores ZooKeeper cair, o cliente receberá uma notificação (*watcher*) avisando que o servidor caiu.

O Apache ZooKeeper foi integrado à plataforma de federação de nuvens ZooNimbus para fornecer uma nova forma de trocar informações entre os servidores e clientes. Com o ZooKeeper, as funções P2P presentes no ZooNimbus foram substituídas pela centralização e coordenação oferecida com os seus recursos.

Com a integração do ZooKeeper, todos os servidores do ZooNimbus consultam dados referentes aos outros servidores dentro do próprio ZooKeeper. Dessa forma, toda vez que um servidor é iniciado dentro da federação, é criado um *znode* com seus principais dados dentro do ZooKeeper, assim sendo, quem for consultar ou atualizar dados dos servidores, fará estas operações dentro do próprio ZooKeeper. Na política de armazenamento proposta, vários dados são obtidos a partir de um *znode* no ZooKeeper para o cálculo do custo de armazenamento de um servidor, e também para o tratamento dos arquivos.

III. Apache Avro

Apache Avro [7] é um sistema de serialização de dados e de chamada remota de procedimentos (RPC - *Remote Procedure Call*) [13] desenvolvido dentro do projeto Hadoop [8] da Apache. Ele é um sistema que oferece:

- Estrutura de dados rica, por meio de esquemas;
- Formato compacto e rápido de dados binários;
- Um *container* de arquivos para armazenar dados persistentes;
- Chamada remota de procedimentos (RPC - *Remote Procedure Call* [13]);
- Integração simples com linguagens dinâmicas, ou seja, não é necessária a geração de código para ler e escrever arquivos de dados, nem para usar ou implementar protocolos RPC. Geração de código funciona como uma opção de otimização, só vale a pena implementar para linguagens de tipagem estática.

Quando os dados do Avro são lidos, o esquema utilizado na escrita deles está sempre presente com os dados. Isso permite que cada dado escrito não tenha despesas de valor, tornando a serialização de dados rápida e pequena, pois os dados, juntos com os seus esboços, são totalmente auto-explicativos, isto facilita o uso de linguagens de *script* dinâmicas. Quando os dados Avro são armazenados em um arquivo, seu esquema é armazenado junto com ele,

de forma que os arquivos possam ser processados por qualquer outro programa futuramente. Se o programa de leitura dos dados espera um esquema diferente, isso pode ser facilmente resolvido, uma vez que ambos os esquemas estão presentes.

Quando o Avro é usado em RPC, o cliente e o servidor usam trocas de esquemas de dados quando uma conexão é estabelecida. Uma vez que o cliente e o servidor têm esquemas do outro, a correspondência entre os mesmos campos nomeados, campos em falta, campos extras, etc, podem ser facilmente resolvidas.

Logo, esquemas Avro são definidos com JSON [6]. Isso facilita a implementação em linguagens que já possuem bibliotecas JSON. Anteriormente, a comunicação entre servidores e clientes no ZooNimbus era realizada por meio de mensagens que eram enviadas pela arquitetura de rede P2P. Esta forma de comunicação foi substituída pelas chamadas RPC do Avro, pois com a comunicação feita com P2P, era necessário um *broadcast*, para descobrir os computadores envolvidos na rede e a mensagem era enviada para todos, já com o RPC a mensagem é enviada diretamente para o computador destino.

Dessa forma, o Avro fornece uma interface de comunicação mais simples e robusta ao ZooNimbus. Como o Avro é um programa que trabalha com recursos de rede para gerar e mandar as RPC, o Avro gerou classes dentro do ZooNimbus que implementam os métodos que são chamados remotamente, tanto por clientes quanto por servidores.

IV. ZooNimbus

Inicialmente, proposta por Saldanha et al. [2] a arquitetura ZooNimbus para federação de nuvens computacionais híbridas, permite a integração e o controle de diferentes provedores de infraestrutura, com suas ferramentas de bioinformática oferecidas como serviço, de maneira transparente, flexível e tolerante a falhas, com grande capacidade de processamento e armazenamento.

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

Para isso, a arquitetura do ZooNimbus é dividida em três camadas, conforme mostrada na Figura 1. A primeira é a camada de infraestrutura, que possui as nuvens que são integradas à arquitetura. Nessa camada, devido a facilidade e dinamismo que o ZooNimbus oferece, a qualquer momento uma nuvem pode ser integrada ou retirada da federação. A segunda é a camada de núcleo, que possui os serviços do ZooNimbus. Atualmente, essa camada é composta por seis componentes, onde cada um oferece um serviço independente do outro, fazendo com que as funcionalidades de cada um sejam bem distintas.

O serviço de monitoramento é responsável por acompanhar a situação de cada servidor da federação. Junto com

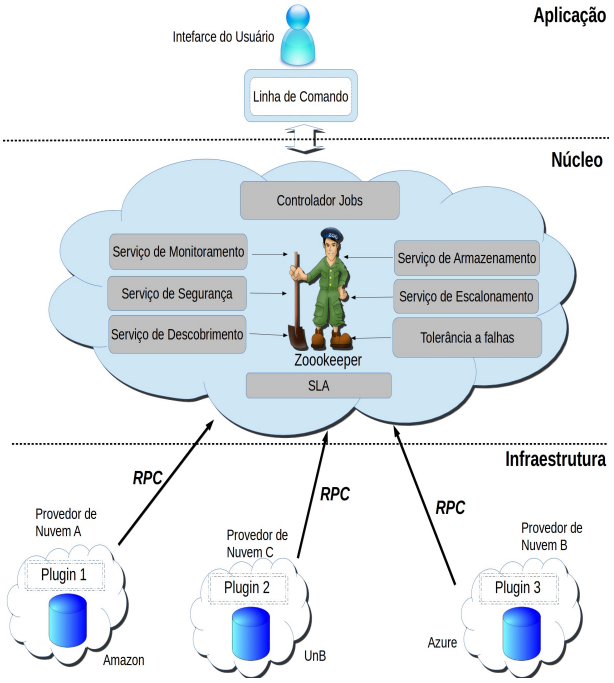

Figura 1. Arquitetura de Federação em Nuvens ZooNimbus.

o ZooKeeper, o serviço de monitoramento avisa todos os servidores quando um dado é alterado em um *znode* e também quando um servidor se desconectou da federação. Todos estes avisos são feitos por meio dos *watchers*. O serviço de segurança faz com que a integração de nuvens ocorra com a segurança estabelecida de acessos, e que isso seja realizado de acordo com os requisitos e políticas internas de todos os componentes da nuvem integrada. O serviço de descobrimento realiza a rotina de descoberta da existência e da indisponibilidade de servidores por meio do ZooKeeper. Como o ZooKeeper utiliza os *watchers* para notificar a indisponibilidade de um servidor, não tem necessidade do serviço de descobrimento realizar verificações de indisponibilidade. O serviço de escalonamento realiza o controle de execução das tarefas no ZooNimbus, para isto, é utilizada uma política de escalonamento para distribuir essas tarefas entre os servidores. O serviço de armazenamento realiza o controle da recuperação e do envio dos dados para a federação, para isto é utilizada a política proposta por este trabalho, ZooClouS (*ZooNimbus Cloud Storage*), explicada na Seção V.

E a terceira é a camada de aplicação, na qual o usuário se conecta a federação por meio de uma interface gráfica ou linha de comando que possibilita o usuário executar suas aplicações na federação de nuvens.

V. *A Política de Armazenamento ZooClouS*

O ZooNimbus, não de forma exclusiva, tem sido usado para executar robustas aplicações de bioinformática. Essas aplicações de um modo geral possuem características semelhantes, as principais são:

- Um conjunto de arquivos de entrada que serão analisados por uma ferramenta, que podem ser originados

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

Estas aplicações podem executar rapidamente se o arquivo de entrada estiver na nuvem que vai realizar o processamento, portanto, uma política que faça com que este arquivo esteja sempre no servidor requisitante melhoraria o desempenho da aplicação. Para melhorar este cenário foi implementada a política de armazenamento de dados ZooClouS.

O ZooClouS realiza vários serviços dentro do ZooNimbus, além de melhorar o desempenho das aplicações, também realiza um controle na forma em que os dados entram, saem e são replicados dentro da federação de nuvens.

Para facilitar o controle dos dados armazenados, o ZooClouS realiza as suas operações em conjunto com o ZooKeeper utilizando os seus *znodes*. Cada servidor que se conecta à federação recebe um identificador único (ID). Após isso, é criado um *znode* no ZooKeeper com este ID. Dentro deste *znode* é criado o *znode STATUS*, onde sua função é informar para o serviço de monitoramento qual a situação daquele servidor. Quando um servidor se desconecta da federação, o *znode STATUS* é mudado para *STATUSWAITING*, isso mostra para o serviço de monitoramento que as rotinas de tolerância a falhas devem iniciar imediatamente para aquele servidor.

O outro *znode* criado é o *files*, nele são criados outros *znodes* filhos que correspondem aos arquivos que um servidor possui. Para cada arquivo é criado um *znode* com os seus dados (ID, nome, tamanho e uma *pluginList*). A *pluginList* é uma lista de todos os servidores que possuem este arquivo, pois o serviço de armazenamento replica todos os dados inseridos na federação. Este processo é mostrado na Seção V-B.

Além destes, também é criado o *znode pending_save*, que recebe dados de um arquivo que deverá ser enviado para a federação. A função da *pending_save* é avisar para todos os servidores que um arquivo deve ser inserido em algum servidor. Para cada arquivo que se deseja enviar a federação, é criado um *znode* dele na *pending_save*, conforme mostrado na Figura 2. Assim, este *znode* é apagado somente depois que o arquivo foi enviado para algum servidor e foi verificado que ele chegou com sucesso.

Com a implementação do ZooClouS, um cálculo de custo de armazenamento foi proposto, e com os resultados deste cálculo é possível escolher o servidor que possui o menor custo de armazenamento para receber o arquivo que foi solicitado o envio.

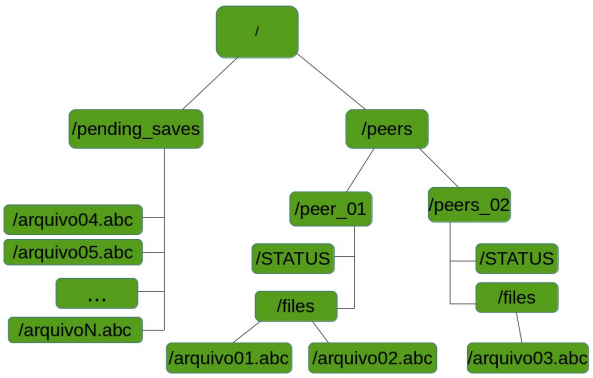

Figura 2. Estrutura do ZooKeeper no ZooNimbus com Relação ao Serviço de Armazenamento.

A. O Cálculo do Custo de Armazenamento

Nesse cenário, quando um arquivo é enviado para a federação de nuvens ou replicado dentro dela, um custo de armazenamento é calculado para aquele arquivo. Este valor é gerado de forma transparente entre o cliente que requisitou a operação de *upload* e cada servidor que tiver espaço em disco suficiente para armazenar o arquivo. Para o ZooNimbus, foram utilizadas três variáveis para o cálculo do custo de armazenamento:

- *Free Size*: espaço livre em disco do servidor;
- *Uptime*: tempo que o servidor está *on-line* na federação, pois um servidor que está há mais tempo *on-line* mostra-se mais confiável de receber o arquivo em comparação a outro que está menos tempo, por ter se desconectado ou por ter se conectado a federação a pouco tempo.
- *Latency*: latência entre o cliente e o servidor de destino, é o maior peso no cálculo do custo.

Na política ZooClouS, o *uptime* tem um peso maior do que o espaço livre no cálculo do custo. Assim, toda vez que um servidor é desconectado da federação, o valor do *uptime* é zerado e a contagem inicia novamente quando ele se reconecta na federação.

O fato de zerarmos o *uptime* não interfere no julgamento de servidores estáveis, baseado nos artigos [15] e [19] que demonstram que o servidores estáveis, se manterão estáveis, mesmo que o *uptime* seja zerado, se uma máquina permanece *online* por uma hora, ela tende a ficar mais uma hora *online*, diferenciando assim dos servidores que caem constantemente.

Logo, a partir dos dados coletados, o cálculo do custo de armazenamento foi criado baseado na proposta de Ren-Xun [20], onde o custo total de armazenamento de um arquivo provém do custo de transferir somado com o custo de armazenar determinado arquivo. O ZooClouS coleta dados necessários para o cálculo do custo de armazenamento dos servidores e define em cada servidor a variável *storagecost*, que será o seu custo de armazenamento de um arquivo de acordo com o cliente que realizou a requisição

de envio. Cada cliente pega uma lista dos servidores para calcular e setar o custo, isto evita que um custo seja sobrescrito por outro cliente antes de ser utilizado pelo cliente anterior. Dessa forma, o custo de armazenamento utilizado na política é uma forma aproximada de definir o tempo de transferência entre dois servidores. Quanto menor o valor, menor será o custo gasto na transferência do arquivo. As três variáveis utilizadas para o cálculo do custo de armazenamento dão origem a Equação 1:

$$S = (U + F) * L \quad (1)$$

Onde S é o custo de armazenamento, U é o tempo que o servidor está *on-line* (*uptime*), F é o espaço livre (*free size*) em disco, e L é a latência calculada entre origem e destino. Além disso, as variáveis utilizadas possuem pesos com o objetivo de priorizar os campos mais importantes no cálculo, assim tem-se a Equação 2:

$$\alpha + \beta + \gamma = 1.0 \quad (2)$$

Com base em vários testes feitos, os valores dos pesos que definiram um melhor custo de armazenamento foram definidos de tal forma que, α com um peso de 0.5 é associado à latência, colocando metade do peso do cálculo nela; β com um peso de 0.2, é associado ao *freesize*; e o γ é atribuido ao *uptime* e possui um peso de 0.3, conforme mostrado na Equação 3:

$$S = (U * \gamma + F * \beta) * (L * \alpha) \quad (3)$$

Esse cálculo forma a primeira parte do custo de armazenamento. Todavia, como o ZooNimbus trabalha com nuvens públicas ou privadas, pode acontecer de uma nuvem ser integrada à federação e ela cobrar um preço pela utilização de seu recurso de armazenamento. Por conta disso, é necessário adicionar ao custo de armazenamento o preço de armazenar um dado em uma nuvem que faça cobrança. No caso de nuvens que, geralmente, pertencem a uma organização privada ou instituição pública e que não cobrem valores para a utilização de seu disco, o preço do seu armazenamento é 0. Para outros casos, como por exemplo, a Amazon [14] ou Azure [5], um valor é cobrado por *gigabyte* de dado armazenado. Este valor, definido em dólares por *gigabyte*, é inserido no arquivo de configuração da nuvem.

Com o valor do preço de armazenagem definido, este custo, representado pela variável *costs*, é somado ao custo de armazenamento calculado daquela nuvem, dando origem a Equação 4:

$$S = (U * \gamma + F * \beta) * (L * \alpha) + Costs \quad (4)$$

Depois de calculado o custo final de cada servidor, uma lista é recebida com todos os servidores e ordenada de acordo com os seus custos de armazenamento. A ordenação é feita em ordem crescente, ou seja, do servidor que tem o menor custo até o maior, sendo que quanto menor o custo, melhor será a transferência de um arquivo para este servidor. Após devolver esta lista para quem está requisitando um envio de *upload*, o cliente irá tentar enviar o arquivo ao primeiro da lista, caso não consiga, continuará tentando o envio no próximo servidor até que o arquivo seja enviado. Além do *upload*, outros serviços foram implementados pela política de armazenamento, ZooClouS, estes são explicados na Seção V-B.

B. Serviços do ZooClouS

Com o objetivo de otimizar o controle dos dados e o seu tratamento dentro de um ambiente de federação de nuvens, o ZooClouS conta com vários serviços:

- Listagem: lista todos os arquivos encontrados na federação. Assim, por meio de um único comando o cliente consegue visualizar todos os arquivos que estão salvos no ZooNimbus. Estes arquivos são mostrados em forma de uma única lista, dando para o usuário a ilusão de que todos estes arquivos estão armazenados somente no servidor que ele está conectado;
- *Download*: *download* de arquivos para o computador do cliente ou para o servidor que precisar de um arquivo para a execução de uma tarefa. Nesse caso, tanto o *download* quanto o *upload* são realizados por meio do protocolo SFTP [11];
- Replicação de todos os arquivos contidos na federação: toda vez que um novo arquivo é enviado ou gerado no ZooNimbus, o ZooClouS verifica quantas cópias existem daquele arquivo armazenadas na federação. Caso o número de cópias seja menor que o valor definido, o arquivo será replicado até atingir o número de cópias, definido no arquivo de configuração. Por padrão estão definidas duas cópias, mas esse valor é facilmente alterado.
- Tolerância a falhas: problemas na rede ou de energia são inevitáveis e pode acontecer de um servidor se desconectar do ZooNimbus, caso isto ocorra, todos os servidores da federação irão receber um aviso através do ZooKeeper, informando que um servidor está inacessível. A partir deste momento dá-se início a rotina de tolerância a falhas, que irá no *znode* do servidor que se desconectou e irá pegar informações sobre os arquivos que se tornaram inacessíveis. Com estas informações, o ZooClouS irá saber quais servidores possuem uma cópia de cada arquivo que estava no servidor que se desconectou. Assim, uma mensagem é enviada para estes servidores avisando que eles devem replicar os arquivos, pois com a queda do servidor no ZooNimbus, possuem no momento uma cópia a menos. Assim, cada servidor que fizer a replicação realiza um cálculo de custo de armazenamento entre ele e todos os outros servidores da federação e então inicia a replicação dos arquivos para o servidor escolhido.

A Figura 3 mostra, de forma simples, como ocorre a utilização do Avro no ZooNimbus no caso de uma operação de *upload*. Como mostrado na Figura 3, (1) o cliente faz uma conexão com um servidor na federação, e (2) recebe uma lista com os servidores disponíveis para receber o arquivo, (3) depois o cliente calcula a latência entre ele e os servidores e envia uma chamada RPC para

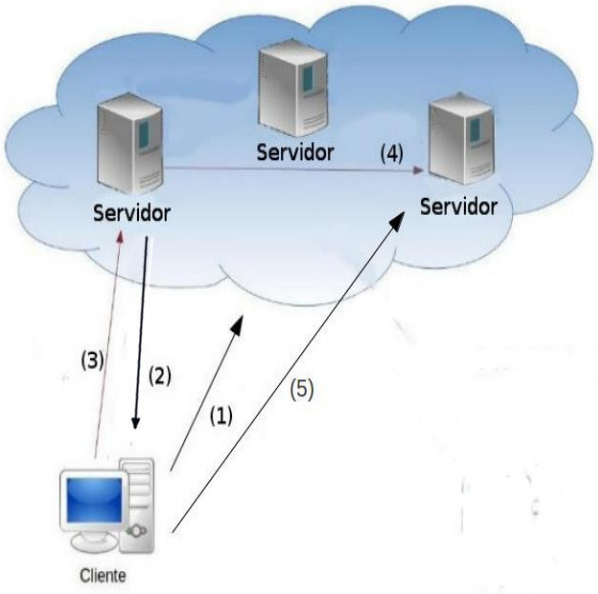

Figura 3. Chamadas RPC Passo-a-Passo.

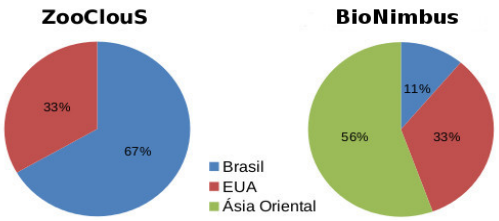

Figura 4. Destino de Arquivo Enviado ao ZooNimbus com a Implementação do ZooClouS, e sem a implementação do ZooClouS.

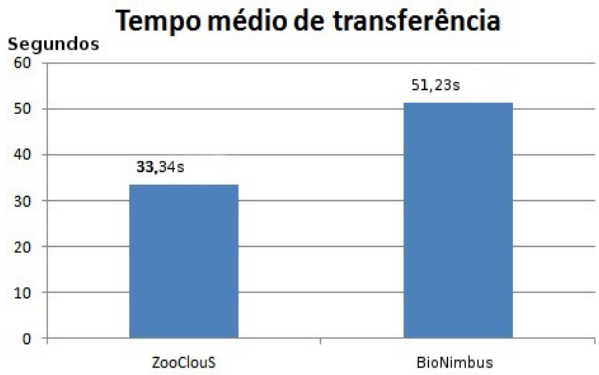

Figura 5. Tempo Médio de Transferência de Arquivo ao ZooNimbus com a Implementação do ZooClouS, e sem a Implementação do ZooClouS.

o servidor em que ele está conectado com os dados que ele gerou e que serão utilizados no cálculo da política de armazenamento. Após esse servidor receber os dados, decide qual o melhor servidor para colocar o arquivo com base nos cálculos da ZooClouS. Em seguida, (4) faz uma chamada RPC para este servidor com os dados do arquivo que o cliente pretende enviar para o ZooNimbus. Após isso, (5) o cliente abre uma conexão com o servidor de destino e envia o arquivo.

VI. Análise dos Resultados

Para os testes da política de armazenamento ZooClouS, foi criada uma federação com três nuvens, as quais são:

- Uma nuvem na Universidade de Brasília, com um processador Core i7-3770 de 3,4 GHz, memória RAM de 8Gb, 2TB de disco rígido, com o sistema operacional Linux, distribuição Ubuntu 13.04. Essa nuvem está fisicamente localizada no Brasil;
- Uma nuvem na Microsoft Azure [5], máquina virtual com 8 núcleos, 14Gb de RAM e disco rígido de 30Gb. Sistema operacional Linux, distribuição Ubuntu 12.04 LTS. A nuvem está localizada na Ásia Oriental;
- Uma nuvem na Amazon [14], máquina virtual com processador Intel Xeon 2.4 Ghz, 613MB de memória RAM, disco rígido de 80GB e sistema operacional Linux, distribuição Ubuntu 12.04.2. A nuvem está localizada nos EUA.

O teste consistiu em realizar uma operação de *upload* para esta federação com o cliente localizado no Brasil, o arquivo utilizado possui 70 *Megabytes* de tamanho, do tipo binário. Foram feitos nove *uploads* de forma aleatória e nove com a política de armazenamento ZooClouS. A forma aleatória é a forma como o BioNimbus realiza operações de *upload*. Ele recebia a requisição e enviava o arquivo para o primeiro servidor que encontrava. Dessa forma, no BioNimbus a escolha deste servidor não possuía nenhum tipo de critério, o servidor era escolhido aleatoriamente, onde qualquer servidor poderia ser escolhido para receber o arquivo. Conforme mostrado na Figura 4, com o ZooClouS, observou-se que em 67% dos envios, o arquivo foi enviado para a nuvem localizada no Brasil. Por outro lado, com a política de armazenamento aleatória, 56% dos envios resultaram na nuvem localizada na Ásia. Isso demonstra que o ZooClouS, com base no cálculo feito, envia o arquivo para o servidor que estiver mais próximo do cliente, pois a latência é menor quando a origem e o destino estão mais próximos geograficamente. Como a política considera a latência como fator mais importante, o resultado do tratamento é o envio do arquivo ao servidor mais próximo possível do cliente.

Assim, com a escolha do servidor de destino mais próximo do cliente, a transferência de dados torna-se mais rápida, conforme mostra a Figura 5. A taxa de transferência média foi calculada com base no tempo médio de transferência do arquivo nas nove operações realizadas em cada algoritmo. Com a implementação do ZooClouS, o tempo de transferência foi minimizado em cerca de 65%. Além disso, observa-se que o tempo médio de transferência para o arquivo foi de 33,34 segundos com o ZooClouS, contra 51,23 segundos da forma aleatória do BioNimbus.

Para realizar as verificações do tempo de execução total das tarefas enviadas para o ZooNimbus, foi criado um conjunto de tarefas. As tarefas executadas para os testes utilizaram a ferramenta Bowtie [12], uma ferramenta de bioinformática, disponível como serviço nos servidores

Tabela I
NOMES E TAMANHOS DOS ARQUIVOS DE ENTRADA DAS TAREFAS.

Nome	Tamanho
hs_alt_HuRef_chr1.fa	252 MB
hs_alt_HuRef_chr2.fa	247 MB
hs_alt_HuRef_chr3.fa	198 MB
hs_alt_HuRef_chr4.fa	189 MB
hs_alt_HuRef_chr5.fa	178 MB

ZooNimbus.

Nesse cenário, foi criado um grupo de tarefas composto por cinco tarefas. Tais tarefas tinham arquivos de entrada de diferentes tamanhos, variando de 178 até 252 MB cada, disponível em ftp://ftp.ncbi.nih.gov/genomes/H_sapiens/Assembled_chromosomes/seq/, banco de dados do NCBI [16]. Os arquivos de entrada, que podem ser visualizados na Tabela I com os seus respectivos tamanhos, estavam presentes nos servidores ZooNimbus.

As tarefas foram enviadas para execução sequencial, e a medida do tempo de execução total foi feita em segundos, onde seu tempo foi calculado entre o envio da primeira tarefa para a execução e o tempo de finalização da última tarefa. Assim, foi enviado para execução o conjunto de tarefas que continham os arquivos de entradas descritos na Tabela I. Para cada arquivo de entrada foi definido um conjunto de tarefas as quais foram agrupadas nas cinco tarefas indicadas na Figura 6.

Para o estudo de caso do tempo de execução de uma tarefa foi criado um ambiente com uma federação que possui duas nuvens, as quais são:

- Uma nuvem na Universidade de Brasília, localizada no Brasil, com três computadores com processadores Core i7-3770 de 3,4 GHz, memória RAM de 8Gb, 2TB de disco rígido, sistema operacional Linux, distribuição Ubuntu 13.04. Nuvem fisicamente localizada no Brasil;
- Uma nuvem na Amazon [14], com três máquinas virtuais que possuem as seguintes configurações: processador Intel Xeon 2.4 Ghz, 613MB de memória RAM, disco rígido de 80GB e sistema operacional Linux, distribuição Ubuntu 12.04.2. Nuvem localizada nos EUA.

Com o ambiente definido, foram realizadas duas execuções, uma com a nuvem localizada na Universidade de Brasília e a outra execução com a federação criada pelas nuvens da Amazon da UnB, ambas com o ZooClouS. Na primeira execução, foi utilizado apenas um servidor ZooKeeper para gerenciar os dados das nuvens. No segundo caso, observou-se durante os testes que com mais servidores ZooKeeper o tempo de escalonamento e execução das tarefas tornou-se mais rápido, portanto foram levantados três servidores ZooKeeper, um na nuvem da UnB e os outros dois na nuvem da Amazon. A Figura 6 mostra os resultados obtidos nos testes realizados.

Na nuvem criada somente na UnB, o tempo total de execução das cinco tarefas foi de 401 segundos. Na integração com a Amazon o tempo total foi de 190 segundos, o que resultou em uma execução 53% mais rápida.

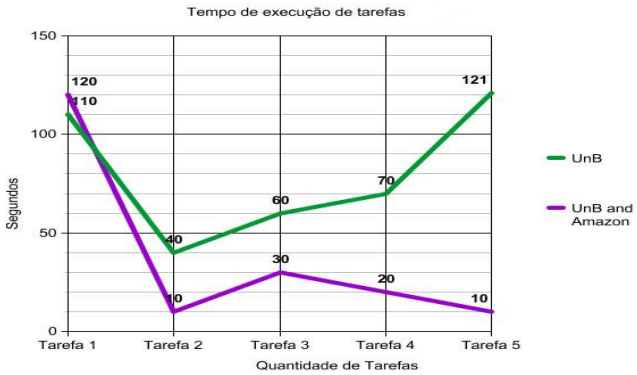

Figura 6. Tempo de Execução de Tarefas com o ZooNimbus.

VII. CONCLUSÃO

Neste trabalho foi proposto o ZooClouS, uma política de armazenamento de dados para a arquitetura de federação em nuvens ZooNimbus. A política ZooClouS buscou realizar um tratamento nos dados armazenados na arquitetura de forma a otimizar o seu desempenho na execução de tarefas. Com base nos testes realizados, concluiu-se que o envio de arquivos à federação de nuvens ZooNimbus está ocorrendo de forma menos custosa ao cliente e ao servidor. Com as operações de replicação e tolerância a falhas implementadas, a disponibilidade dos dados aumentaram e também a obtenção de arquivos pelo cliente e pelo servidor, com a operação de *download*.

Para trabalhos futuros propõe-se a implementação de uma lista de controle de acesso (ACL) para ser associada aos arquivos inseridos na federação, de modo que somente usuários autorizados possam utilizar e modificar tais arquivos. Além disso, propõe-se uma implementação do protocolo P-FTP (*Parallelized File Transfer Protocol*) [18] para um ambiente de federação em nuvens, de forma a aumentar a velocidade na transferência de dados entre os clientes e os servidores. Apesar dos bons resultados obtidos com a fórmula atual, observamos a necessidade de modificá-la, mantendo as premissas e parâmetros atuais para que possamos no futuro obter resultados ainda melhores.

VIII. AGRADECIMENTOS

Este trabalho está sendo apoiado pelo STIC-AmSud (CAPES).

REFERÊNCIAS

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[2] C. A. L. Borges, Hugo Saldanha, Edward de Oliveira Ribeiro, Maristela Holanda, Aleteia Araujo, and Maria Emilia M. T. Walter. Task scheduling in a federated cloud infrastructure for bioinformatics applications. In Frank Leymann, Ivan Ivanov, Marten van Sinderen, and Tony Shan, editors, *CLOSER*, pages 114–120. SciTePress, 2012.

[3] R. Buyya, R. Ranjan, and R. N. Calheiros. Intercloud: utility-oriented federation of cloud computing environments for scaling of application services. In *Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I*, ICA3PP'10, pages 13–31, Berlin, Heidelberg, 2010. Springer-Verlag.

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

[5] Microsoft Corporation. Windows azure. http://www.windowsazure.com/pt-br/pricing/free-trial/, 2012.

[6] Douglas Crockford. Json). http://json.org/, 2012.

[7] Apache Software Foundation. Apache avro). http://avro.apache.org/, 2012.

[8] Apache Software Foundation. Apache hadoop). http://hadoop.apache.org/, 2012.

[9] The Apache Software Foundation. Apache zookeeper. http://zookeeper.apache.org/, 2010.

[10] Google. Google app engine. https://developers.google.com/appengine/docs/whatisgoogleappengine?hl=pt-br, 2013.

[11] JCraft. Pure implementation of sftp for java. http://www.jcraft.com/jsch/examples/Sftp.java.html, 2012.

[12] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg. Ultrafast and memory-eficient alignment of short dna sequences to the human genome, 2009.

[13] Kwei-Jay Lin and J.D. Gannon. Atomic remote procedure call. *Software Engineering, IEEE Transactions on*, SE-11(10):1126–1135, 1985.

[14] Amazon Web Services LLC. Amazon elastic compute cloud (EC2). http://aws.amazon.com/pt/ec2/, 2012.

[15] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, pages 53–65, London, UK, UK, 2002. Springer-Verlag.

[16] U.S. National Library of Medicine. National center for biotechnology information. http://www.ncbi.nlm.nih.gov/, 2013.

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

[18] S. Sohail, Sanjay Jha, and H. ElGindy. Parallelized file transfer protocol (p-ftp). In *Local Computer Networks, 2003. LCN '03. Proceedings. 28th Annual IEEE International Conference on*, pages 624–631, 2003.

[19] H. Verespej and J. Pasquale. A characterization of node uptime distributions in the planetlab test bed. In *Reliable Distributed Systems (SRDS), 2011 30th IEEE Symposium on*, pages 203–208, 2011.

[20] Ren Xun-Yi and Ma Xiao-Dong. A* algorithm based optimization for cloud storage. *JDCTA*, 4(8):203–208, 2010.