AcoSched - Scheduling Algorithm in a Federated Cloud Infrastructure for Bioinformatics Applications

Gabriel Silva Souza de Oliveira, Edward Ribeiro, Diogo Assis Ferreira
Aletéia P. F. Araújo, Maristela T. Holanda, Maria Emilia M. T. Walter
Dept. of Computer Science
University of Brasília - UnB
Distrito Federal
Email: (gabriel_222zt, edward.ribeiro, diogo.assis.ferreira)@gmail e (aleteia, mholanda, mia)@cic.unb.br

***Abstract*—Task scheduling is difficult in federated cloud environments, since there are many cloud providers with distinct capabilities that should be addressed. In bioinformatics, many tools and databases requiring large resources for processing and storing enormous amounts of data are provided by physically separate institutions. This article treats the problem of task scheduling in ZooNimbus, a federated cloud infrastructure for bioinformatics applications. We propose a scheduling algorithm based on Load Balancing Ant Colony (LBACO), called AcoSched, to perform an efficient distribution for finding the best resources to execute each required task. We developed experiments with real biological data executing on ZooNimbus, formed by some cloud providers executing in Amazon EC2 and UnB. The obtained results show that AcoSched makes a significant improvement in the makespan time of bioinformatics applications executing in ZooNimbus, when compared to the Round Robin algorithm.**

I. Introdução

Atualmente, computação em nuvem é vista como uma realidade das plataformas distribuídas que proporciona um vasto cenário de pesquisas e novas propostas de estudos acadêmicos e científicos. Celesti *et al.* [3] preveem uma evolução para a computação em nuvem que soluciona alguns problemas existentes neste ambiente, tal como a possível escassez de recursos em uma nuvem, que acaba não sendo capaz, em algum determinado momento, de suprir a demanda e atinge um limite físico de expansão. A evolução prevista por Celesti *et al.* consiste em três etapas: a etapa *"Monolithic"*, na qual os serviços de nuvens estão baseados em arquiteturas proprietárias, ilhas de nuvens que disponibilizam serviços sem o compartilhamento de recursos entre diferentes provedores de nuvens; a etapa *"Vertical Supply Chain"*, onde as nuvens ainda são ilhas proprietárias, mas a utilização de outros provedores de recursos já começa a ocorrer; e a etapa *"Horizontal Federation"*, onde os diversos provedores de nuvens irão cooperar e utilizar os recursos de outros provedores, gerando economia de escala, uso eficiente de seus recursos e aumento de sua capacidade [3]. O presente trabalho utiliza o conceito da terceira etapa da evolução de nuvens computacionais, *"Horizontal Federation"*, conhecida também como nuvem federada, que é disponibilizada por meio de um *middleware*. The middleware provides the functionalities that enable clouds belonging to different organizations, and subject to different policies, to be shared in a consistent and controlled way.

Assim, o objetivo deste artigo é propor um algoritmo de escalonamento dinâmico para o ambiente de nuvens federadas, que minimize o *makespan* da execução de uma tarefa, e realize o balanceamento de carga. Para isso, o algoritmo proposto considera a localidade dos arquivos de entrada ao realizar o escalonamento e os tipos de serviços solicitados. O algoritmo de escalonamento proposto foi chamado de AcoSched e é baseado na política de escalonamento LBACO (*Load Balancing Ant Colony*) [11], a qual utiliza a heurística de colônia de formigas. Para a implementação do AcoSched foi necessária a utilização de um serviço de coordenação no ambiente de nuvens federadas e a mudança na forma de comunicação entre os recursos que integram a federação. Essas mudanças levaram a criação de uma nova versão do BioNimbus, chamada de ZooNimbus [16], o qual proporcionou um ambiente mais escalável, dinâmico e de fácil programação, e que será descrito na Seção IV.

Para isso, este artigo foi estrutura em mais sete seções. A Seção II apresenta as principais características da bioinformática que se beneficiam de um ambiente de nuvem federada. A Seção III apresenta alguns trabalhos relacionados. Na Seção IV são descritos os detalhes do ambiente de federação em nuvem, ZooNimbus, usado neste trabalho. Em seguida, na Seção V é apresentado o algoritmo de escalonamento proposto para federação em nuvem. Para finalizar, as Seções **??** e VII mostram os resultados obtidos a partir de dados reais e algumas conclusões, respectivamente.

II. Bioinformática

III. Trabalhos relacionados

Nesta seção é apresentada uma explanação das funcionalidades apresentadas por alguns algoritmos de escalonamento para nuvens federadas, analisando suas principais características e diferenças. Quan Chen et al. [4] apresentam o algoritmo SAMR (a Self-Adaptive MapReduce scheduling algorithm) para escalonar tarefas do modelo de programação MapReduce, que é capaz de detectar dinamicamente quais são os jobs que consomem mais tempo de processamento da nuvem e faz com que eles sejam executados apenas depois que os jobs rápidos já tenham sido finalizados.

Por outro lado, o artigo [6] é focado em tarefas especificas de um ambiente de e-learning, apresentando as características deste tipo de sistemas, suas necessidades e como um bom escalonador deve trabalhar neste contexto.

Yee-Ming Chen et al. [5] desenvolveram um algoritmo de reescalonamento baseado em heurística (HEFT), no qual

após escalonar algum job o escalonador pode perceber que a máquina na qual o job está sendo executado caiu ou está sobrecarregada e, então, realizar um re-escalonamento para um outro nó que esteja plenamente funcional e não sobrecarregado. O algoritmo utiliza um repositório de histórico da performance como base para aprimorar as estimativas subsequentes.

No artigo [19] é proposto o algoritmo PISA (Priority Impact Scheduling Algorithm), o qual estende as características do algoritmo FIFO (First In, First Out) para levar em consideração a prioridade que o usuário dono das tarefas possui. Sendo que essa prioridade pode ser determinada pelo valor pago pelo usuário, ou se ele é ou não um administrador da rede, por exemplo.

Em [12], foi proposto um algoritmo de otimização de balanceamento de carga baseado em colônia de formigas (LBACO). Ele é capaz de encontrar a alocação de recurso ótima, além de balancear a carga da nuvem durante o processo. O escalonador se adapta ao modelo dinâmico que a computação na nuvem possui.

O problema de compartilhamento de arquivos entre jobs no paradigma da computação nas nuvens é trabalhado no artigo [14], o qual propõe um algoritmo de escalonamento que leva isso em consideração, o CSA (Cloud Sharing Aware).

Assim, na próxima seção esses artigos serão avaliados e classificados, de acordo com suas características no paradigma da computação em nuvens.

A. Avaliação das Abordagens

Os artigos previamente apresentados foram avaliados tendo em mente as características fundamentais para definir um bom escalonador. Para esta análise foi usado como base o trabalho de Stelios et al. [17], no qual são julgados como características importantes para um escalonador: Flexibilidade, Escalabilidade, Heterogeneidade, Balanceamento de carga, Exposição de informação, Dados em tempo real, Histórico de escalonamento, Conservação de energia, Analise da banda e Prioridade.

Flexibilidade determina se o algoritmo pode ser implementado em diferentes arquiteturas sem grandes alterações. A Escalabilidade está relacionada com o fato de o sistema poder crescer sem correr risco de comprometer o desempenho do algoritmo. Heterogeneidade representa se o algoritmo está pronto para lidar em um ambiente computacional heterogêneo. Balanceamento de carga informa se o algoritmo está preocupado em não sobrecarregar um nó enquanto existem outros ociosos. Exposição de informação determina se o escalonador expõem informações sigilosas da nuvem ou das tarefas do usuário. Dados em tempo real diz respeito se o algoritmo está preparado para trabalhar em um ambiente dinâmico. Histórico de escalonamento refere-se ao fato do escalonador armazenar um histórico com as decisões tomadas anteriormente, e se a escolha foi bem sucedida, utilizando esse histórico para auxiliar em escalonamento futuros. Conservação de energia determina se o algoritmo preocupa-se com o consumo de energia elétrica que será gasto para executar o job. Analise de banda verifica se o algoritmo leva em consideração na hora de escalonar, se a rede está congestionada ou não. Por último, Prioridade informa se o escalonador está preparado para trabalhar com tarefas que tenham diferentes níveis de prioridade.

A Tabela I apresenta o resumo das características supracitadas suportadas ou não pelos algoritmos analisados. Na tabela, as colunas representam as características detalhadas anteriormente, enquanto que as linhas apresentam os algoritmos. Nessa tabela, ND significa "Não discutido" no artigo, ou seja, é uma característica que não foi tratada na implementação do artigo; "Sim" significa que o artigo tratou claramente a característica julgada; e "Não" simboliza que o artigo não tratou nenhuma questão sobre a característica. Assim, como pode ser notado na Tabela I, a abordagem apresentada em [4] economizou os recursos computacionais da nuvem nos casos testados pelos autores. Entretanto, ele ainda precisa ser testado em ambientes reais para saber se a otimização será mantida. Além disso, possui outra dificuldade que é encontrar os valores certos dos parâmetros iniciais para uma configuração adequada para cada sistema.

Por outro lado, tudo que é apresentado em [6] é pensado exclusivamente para um ambiente de e-learning, consequentemente, o que é apresentado no artigo não é útil para sistemas com objetivos mais gerais, e nem para bioinformática. Entretanto, para um sistema de e-learning são apresentados métodos interessantes, porém eles são métodos estáticos, o que é uma desvantagem se levado para o conceito de nuvem. Devido a isso, na Tabela [**?**] é explicitado que o escalonador não trabalha com dados em tempo real.

Contudo, a grande vantagem apresentada no algoritmo proposto em [5] é o fato de ele implementar reescalonamento, ou seja, se o sistema perceber que a escolha inicial de escalonamento não está sendo efetiva para uma dada tarefa, a instância do re-escalonador pode distribuir novamente a tarefa, porém desta vez para outra máquina. Entretanto, para essa abordagem funcionar toda a arquitetura da nuvem precisa ser reformulada para a apresentada no artigo.

Em [12] foi apresentado um eficiente algoritmo para escalonamento em nuvem baseado em colônia de formigas que é capaz de balancear a carga entre os nós independente do tamanho das tarefas. Entretanto, o algoritmo não está pronto para lidar com tarefas muito distintas entre si. O real foco do trabalho presente em [19] foi criar um algoritmo de escalonamento para nuvem que levasse em consideração a prioridade do usuário na hora de escolher qual tarefa será escalonada primeiro, devido a isso, o escalonador proposto acaba sendo bem simples nas outras características. Todo o trabalho apresentado em [14] é para tratar o caso em que tarefas distintas precisem compartilhar um mesmo arquivo armazenado na nuvem. Então, ele é adequado para um sistema em que são armazenados muitos arquivos e eles sejam acessados com frequência.

Assim sendo, a partir da análise crítica realizada nos algoritmos citados, este trabalho propõe na Seção IV um algoritmo de escalonamento dinâmico para plataformas de nuvens federadas que trata os problemas levantados nos algoritmos anteriores. Para isso, na próxima seção, é apresentado o ambiente de nuvem federada usada na implementação do algorotmo proposto.

IV. Ambiente ZooNimbus

A política de escalonamento apresentada por este trabalho foi, inicialmente, desenvolvida para a plataforma Bio-

Table I. COMPARAÇÃO ENTRE OS ALGORITMOS

Artigos	CSA[14]	PISA[19]	LBACO[12]	HEFT[5]	For E-learning[6]	SAMR[4]
Flexibilidade	NÃO	ND	SIM	NÃO	ND	ND
Escalabilidade	Sim	ND	SIM	ND	SIM	SIM
Heterogeneidade	ND	SIM	SIM	SIM	SIM	SIM
Balanceamento de Carga	ND	ND	SIM	ND	SIM	ND
Exposição de Informação	ND	ND	ND	NÃO	ND	ND
Dados em Tempo Real	SIM	SIM	SIM	SIM	NÃO	SIM
Histórico de Escalonamento	NÃO	NÃO	SIM	NÃO	ND	SIM
Conservação de Energia	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO
Analise da Banda	NÃO	NÃO	ND	NÃO	SIM	ND
Prioridade	NÃO	SIM	NÃO	SIM	NÃO	NÃO

Nimbus [18], com a finalidade de disponibilizar outra opção de escalonamento, já que o BioNimbus contava com uma política de escalonamento proposta por Borges [18]. Todavia, notou-se a necessidade de obter uma federação mais escalável, dinâmica, de fácil programação e manutenção. Assim, optou-se por manter os módulos de serviço iniciais do BioNimbus proposto por Saldanha [18], mas alterar seu mecanismo de comunicação e coordenação.

A comunicação do BioNimbus era implementada por meio da troca de mensagens P2P, e o controle de toda a nuvem federada era concentrado no núcleo de cada servidor BioNimbus. A descoberta dos recursos no BioNimbus era feito através da rede P2P formada entre eles. Na nova versão do BioNimbus, chamada ZooNimbus, a forma de comunição entre os recursos da federação é realizada por Chamada de Processos Remotos (RPC), e houve a descentralização do controle da federação com a utilização de uma ferramenta de coordenação para ambientes de sistemas distribuídos, o Zookeeper [9].

O ZooNimbus, visualizado na Figure 1, assim como o BioNimbus, é estruturado em três camadas, as quais são aplicação, núcleo e infraestrutura. Na camada de núcleo estão os serviços responsáveis por garantir a integração entre as nuvens na federação. Dentre os serviços implementados na camada de núcleo, os que estão relacionados diretamente com a política de escalonamento proposta neste trabalho são: o Serviço de Escalonamento, que coordenam as tarefas enviadas para execução; o de Serviço de Armazenamento, gerencia os arquivos pertencentes a federação; o Serviço de Monitoramento, que auxilia na garantia de realização das atividades e monitora os recursos que formam a federação; o Serviço de Descoberta e o de Tolerância a Falhas presente nos demais módulos e que garantem o bom funcionamento da plataforma.

Além disso, a comunicação no ZooNimbus foi implementada via RPC, por meio do Apache Avro [7], um sistema de RPC e de serialização de dados desenvolvido pela fundação Apache [8], que foi escolhido dentre as demais tecnologias que promovem a comunicação remota, pela sua flexibilidade na implementação, facilidade de manipulação e o suporte a grande volumes de dados.

Figure 1. Arquitetura da Plataforma de federação ZooNimbus.

Para a descentralização, controle e coordenação da plataforma ZooNimbus, foi implementada a ferramente Zookeeper [9], um serviço de coordenação de aplicações distribuídas, criado também pela Apache [8], para ser de fácil programação e manipulação. Ele utiliza um modelo de dados parecido com a estrutura de árvores de diretórios comum, e possui uma implementação de serviços para a sincronização, a manutenção e a configuração de grupos e nomenclaturas. Tem a finalidade de reduzir a dificuldade de implementar os serviços dos sistemas distribuídos, que podem levar a uma alta complexidade, e a uma grande dificuldade de gestão, de coordenação e de manutenção. O Zookeeper disponibiliza seu serviço como um servidor que armazena dados que são constantemente acessados pelas aplicações distribuídas, esses dados são armazenados nos *znodes*, nós da estrutura persistidas no Zookeeper que pode conter valores de tamanho, no máximo 1 MB, e devem ser voltados para informações de uso geral de controle e coordenação, tais como metadados, caminhos,

endereços, dados de configuração, etc. A Figure 2 mostra a estrutura e a organização dos *znodes* do ZooNimbus.

Figure 2. Estrutura do ZooNimbus no Zookeeper formada por Znodes.

Na Figure 2 é possível observar que todos os *znodes* são referentes a algum módulo de serviço e auxiliam na coordenação da infraestutura de nuvem federada. Como é o exemplo do *znode STATUS*, que sinaliza para todos os módulos a disponibilidade de um recurso, pois quando um novo recurso inicia sua participação na federação, sua disponibilidade é sinalizada pela criação do *znode STATUS* e, por este ser efêmero, irá existir até que este recurso perca sua conexão com o servidor Zookeeper e, consequentemente, deixe de participar da federação. Quando um recurso de uma nuvem estiver executando uma instância do ZooNimbus e se registrar no servidor Zookeeper, ele passa a ser chamado de servidor ZooNimbus e integra a nuvem federada.

Dentre os serviços presentes no ZooNimbus, citados acima e visualizados por meio da Fig.1, o serviço de escalonamento é que realiza o envio da execução de uma tarefa, sendo o responsável pelo controle do escalonamento e execução da mesma.

A. Serviço de Escalonamento

Este serviço é o responsável pela execução de uma tarefa enviada para o ZooNimbus, garantindo que todas as etapas necessárias para o escalonamento e execução funcionem de forma correta. Ele acompanha uma tarefa, antes, durante e após a sua execução.

Quando um cliente está conectado a um servidor ZooNimbus, ou seja, conectado a qualquer nuvem que participe da federação, e solicita a execução de uma tarefa, o servidor será o responsável por receber essa solicitação e lançá-la no *znode jobs* do servidor Zookeeper, juntamente com todas as informações, referentes ao *job*, necessárias para realizar a sua execução. Ao realizar esse lançamento todos os módulos de escalonamento dos *N* servidores ZooNimbus participantes da federação, irão receber uma notificação, alertando sobre o novo *job* a ser escalonado. O método responsável pelo tratamento de notificações do módulo de escalonamento irá recuperar todas as informações lançadas no Zookeeper e criar um *znode BLOCK* para bloqueio desse *job* no servidor Zookeeper, impossibilitando que dois recursos diferentes o encaminhem para a política de escalonamento. Um novo *znode task_IdJob* é criado dentro do *znode task* do recurso eleito para o escalonamento e o *znode job_IdJob*, que pertencia ao *znode jobs* e representava a tarefa antes de ser escalonada é apagado. A criação do *znode job_IdJob* gera um disparo para o recurso responsável por ele, informando-o da existência da nova tarefa que deve ser executada.

Após a execução da tarefa, seu *status* é alterado no servidor ZooNimbus e, novamente, um disparo é feito pelo Zookeeper alertando sobre a finalização da tarefa que o serviço de escalonamento irá tratar para que as devidas rotinas de finalização de tarefas sejam realizadas. O mesmo ocorre para uma tarefa que está em execução e para uma tarefa que foi iniciada a execução, mas ocorreu algum erro.

Neste contexto fez-se necessário implementar um novo algoritmo de escalonamento que fosse capaz de escolher dinamicamente os melhores provedores da nuvem para executar cada tarefa submetida pelo usuário. O algoritmo proposto neste trabalho é detalhadamente descrito na próxima seção.

V. Política de Escalonamento AcoSched

Um bom escalonador deve adaptar sua estratégia de acordo com o ambiente e os tipos de tarefas escalonadas [11]. Considerando essa característica, este trabalho tem o objetivo de criar um escalonador adaptável para a plataforma ZooNimbus. E para tal, foi necessária a coleta de requisitos que deveriam ser observados pela política de escalonamento, concluindo-se que a nova política levasse em conta os fatores: balanceamento dinâmico, capaz de tomar a decisão em tempo de execução da aplicação; a consideração do tipo de recurso sobre o qual o cliente quer executar sua tarefa, se é em recursos pagos ou gratuítos, poupando custos finais; a consideração da capacidade da memória primária disponível, otimizando o tempo de execução das tarefas; a consideração sobre a capacidade computacional total e a disponível no momento do escalonamento; e a consideração da localidade dos arquivos de entrada, minimizando o *overhead* com a transferência dos arquivos e otimizando o tempo total de execução de uma tarefa.

Esses fatores são utilizados pelo escalonador AcoSched para decidir onde atribuir cada tarefa. Essa análise é feita antes da atribuição e durante a execução, pois se for necessário a tarefa poderá ser re-escalonada para um outro recurso.

A análise realizada é traduzida em números, os quais são usados para listar os melhores recursos disponíveis na federação. Essa análise é baseada no algoritmo LBACO [11] e adaptados para o ambiente do ZooNimbus considerando as tarefas de bioinformática que são executadas.

O escalonamento de uma tarefa pelo Acosched, irá considerar o melhor recurso para aquela determinada tarefa, naquele determinado momento do escalonamento, ou seja, ele irá considerar a utilização atual da capacidade computacional de um recurso, procurando sempre aquele mais disponível.

O AcoSched é um algoritmo de busca aleatória que imita o comportamento de uma colônia de formigas real em busca de alimentos utilizando feromônios[1] para traçar os caminhos. Esse feromônio é utilizado de forma análoga pelo AcoSched para sinalizar qual o melhor recurso visitado, podendo sua intensidade ser alterada em cada recurso, o que deve aumentar ou diminuir sua probabilidade.

A escolha de um recurso pelo AcoSched é definida pelo cálculo da probabilidade de escolha de cada recurso visitado. Essa probabilidade é calculada por meio da aleatoriedade de escolha do recurso, da definição do feromônio local, fator que realiza uma escala de qualidade, da capacidade computacional, do balanceamento de carga e da quantidade de memória primária disponível.

Desta forma, para realizar o cálculo da capacidade computacional de um servidor ZooNimbus, valor que será conside-

[1]Feromônio: substância biologicamente muito ativa, secretada esp. por insetos e mamíferos, com funções de atração sexual, demarcação de trilhas ou comunicação entre indivíduos, ferormônio (Etimologia: grego “trazer” sob a f. fero+ hormônio.) [10]

rado ao definir a sua probabilidade de escolha, é utilizada a Equação 1 no instante de tempo t = 0, e com a frequência do processador dada em milhões de instruções por segundo (MIPS).

$$\tau_{ij}(0) = num\_processadores * fq\_processador - lat\_VM \quad (1)$$

O número de processadores é utilizado de forma dinâmica e com uma aproximação máxima do uso real e atual da CPU de um recurso, ou seja, quando é obtido o número de processadores para aplicar na Equação 1, é obtido o número de processadores disponíveis em cada recurso para aquele intervalo de tempo em que é realizado o cálculo da capacidade computacional. Isso pode reduzir o valor da sua capacidade computacional, e aumentar o balanceamento de carga de trabalho, já que, consequentemente, sua probabilidade de escolha dada pela Equação 2, será proporcional ao uso de seus processadores.

A latência da rede é utilizada como fator para medir a capacidade computacional, pois o tempo de transferência do arquivo na rede é determinante para alteração do tempo total de execução. Como, quanto maior a latência, maior pode ser o seu tempo de execução, ela é medida como um fator negativo na capacidade computacional de um servidor ZooNimbus. A latência é calculada entre os recursos, de acordo com a localidade do arquivo desejado e as localidades dos recursos que podem ser eleitos pelo escalonador. O algoritmo inicia colocando as formigas aleatóriamente em alguns servidores ZooNimbus para que elas possam definir a probabilidade de escolha daquele servidor e, então, visitar todas os demais. Após as visitas serem completadas, elas atualizam o valor do feromônio, e se a melhor solução não for encontrada, ou seja, todos os recursos foram visitadas e calculado sua probabilidade de escolha, é realizado novamente as visitas das formigas nos recursos. As iterações das visitas acabam quando a melhor solução é encontrada ou quando o número máximo de iterações é atingido.

A Equação 2, do cálculo da probabilidade de escolha de um recurso, considera os fatores como quantidade de memória primária, capacidade computacional, balanceamento de carga e intensidade do feromônio local. No caso do feromônio inicial, onde um recurso é visitado pela primeira vez no instante t=0, o feromônio será dado também pela Equação 1.

$$p_{ij}^{k}(t) = \frac{[\tau_j(t)]^{\alpha}[EV_j]^{\beta}[LB_j]^{\gamma}[MR_j]^{\delta}}{\Sigma[\tau_k(t)]^{\alpha}[EV_k]^{\beta}[LB_k]^{\gamma}[MR_k]^{\delta}} \quad (2)$$

Na 2, MR_j se refere a capacidade de memória, dada pelo tamanho do espaço livre em memória, e γ é o seu coeficiente de controle para a VM_j. E EV_j representa a capacidade computacional no recurso dada pela Equação 1.

E também onde LB_j representa o balanceamento de carga da VMj, dado pela Equação 3, e para um tempo esperado medido pela Equação 4:

$$LB_j = 1 - \frac{t\_esperado_j - ultimo\_tempo_j}{t\_esperado_j + ultimo\_tempo_j} \quad (3)$$

$$t\_esperado_j = \frac{tam\_total\_tarefa}{EV_j} + \frac{tam\_arquivo\_entrada}{EV_j} \quad (4)$$

Os parâmetros α , β e γ são para controle do peso relativo da trilha do feromônio, da capacidade computacional e do balanceamento de carga das VMs, respectivamente.

Para que o escalonamento dinâmico funcione com a atualização do feromônio marcando o melhor recurso, é necessário que sua intensidade diminua quando estiver com muita carga, deixando de ser o melhor recurso. Para isso, é utilizado uma equação de atualização do feromônio, dada pela Equação 5, onde p é o coeficiente de decaimento do feromônio.

$$\tau_j(t+1) = (1-p) * \tau_j(t) + \Delta\tau_j \quad (5)$$

$$\Delta\tau_j = 1/T_{ik} \quad (6)$$

Quando uma formiga completa suas visitas, o $\Delta\tau_j$ é dado pela Equação 6. Com T_{ik} sendo a menor probabilidade calculada pela formiga na iteração *i*.

Quando a formiga completa suas visitas e encontra a melhor solução, é utilizado $\Delta\tau_j = D/T_{op}$, onde T_{op} é a melhor solução atual, e D é o coeficiente de encorajamento.

Os parâmetros explicitamente utilizados na Equação 2 para definir a probabilidade de escolha do próximo recurso a ser visitado não foram os únicos utilizados para realizar o escalonamento das tarefas. Para atender aos requistos inicias e obter uma boa política de escalonamento, é realizado também um filtro dos recursos disponíveis para verificar qual deles contém os serviços, e qual deles é público ou privado de acordo com a escolha do cliente que solicitou a execução da tarefa.

Os dados utilizados para a execução das equações do algoritmo são armazenados e recuperados como metadados no servidor Zookeeper. Eles são armazenados na forma da estrutura de dados de vetor, e contém dados da última probabilidade de escolha, feromônio, valores dos coeficientes de controle, tamanho do último *job* executado, tempo da última execução e tamanho total das tarefas executadas no recurso.

VI. Resulados

Os testes apresentados nesta seção foram realizados com dados reais de bioinformática, em um ambiente de nuvem federada formado por duas nuvens, as quais foram: uma nuvem na Universidade de Brasília (UnB) e uma nuvem no provedor Amazon EC2. Os testes foram realizados com a política AcoSched e com a política implementada inicialmento no BioNimbus, a DynamicAHP [2].

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 [1], uma ferramenta de bioinformática, disponível como serviço nos servidores ZooNimbus.

Nesse cenário, foi criado um grupo de tarefas composto por 10 tarefas. Tais tarefas tinham arquivos de entrada de diferentes

tamanhos, variando de 178 até 252 MB cada, disponível na internet na página do banco de dados do NCBI [15].

As tarefas foram enviadas para execução sequencial, e a medida do tempo de execução total foi feita em milissegundos, onde seu tempo foi calculado entre o envio da primeira tarefa para a execução e o tempo de finalização da última tarefa. Para cada arquivo de entrada, havia um par de tarefas, totalizando 10 tarefas para um conjunto com cinco arquivos de entrada. O grupo de tarefas criado foi enviado para execução seis vezes, resultando num total de 60 tarefas escalonadas para cada algoritmo avaliado, o DynamicAHP e o AcoSched. Apesar destas tarefas terem sido enviadas sequencialmente, o grupo de tarefa era subdividido em pares e ao enviar um tarefa para execução, o seu par somente era enviado após a confirmação de sua execução.

Os parâmetros de configuração do AcoSched ficaram definidos nos valores mostrados na Tabela II.

Table II. PARÂMETROS DE CONFIGURAÇÃO DEFINIDOS PARA O ACOSCHED.

Parâmetros	Valores
Números de tarefas	10
Números de formigas na colônia	2
Número de iterações	10
p	0.08
α	2
β	4
γ	2

Os resultados dos escalonamentos das tarefas realizadas no ZooNimbus, ao executar o algoritmo de escalonamento AcoSched, foram comparados aos resultados obtidos ao executar a política de escalonamento DynamicAHP [2], e podem ser visualizados na Figura 3.

Figure 3. *Makespan* do Conjunto de Tarefas Executadas na Federação, com duas nuvens.

Na Figura 3, tem-se a analise do tempo total de execução das tarefas no ambiente de nuvem federada ZooNimbus, utilizando-se da nuvem na Universidade de Brasília e da nuvem pública provida pela Amazon [13]. É possível analisar por meio da Figura 3 a diferença do tempo de execução do AcoSched quando comparado ao DynamicAHP, ao executar o algoritmo na nuvem federada, isso porque ao analisar a saída do escalonamento é possível verificar que o DynamicAHP realizou uma maior distribuição das tarefas entre os servidores ZooNimbus, utilizando um maior número de servidores diferentes para executar o conjunto de tarefas em relação ao AcoSched. Essa distribuição, porém, não foi a melhor decisão, visto que alguns desses servidores levaram um tempo maior para executar as tarefas em relação aos demais.

Ao analisar outros resultados obtidos a partir da Figura 4, que mostra os tempos de escalonamentos obtidos, com as primeiras 4 tarefas executadas, visualizadas na Figura 3, nota-se que o AcoSched obtém tempos mais elevados para escalonar uma tarefa do que o DynamicAHP, porém esses tempos de escalonamento não foram refletidos no tempo total de execução das tarefas, o que significa que o AcoSched demora mais para tomar uma decisão ao realizar o escalonamento, porém essa demora é compensada pelo fato da sua decisão resultar em um melhor escalonamento e, consequentemente, um menor tempo total de execução das tarefas.

Figure 4. Tempo de Escalonamento do Primeiro Conjunto de Tarefas das Políticas Analisadas.

O AcoSched mostrou-se eficiente quando comparado com o algoritmo implementado anteriormente no ambiente de federação em nuvem, reduzindo o tempo total de execução dos conjuntos de tarefas ao considerar mais fatores para realizar o escalonamento. Apesar do aumento no tempo de decisão do escalonamento, o algoritmo demonstrou sua eficiência ao compensá-lo no tempo de execução da tarefa, otimizando, assim, as execuções de tarefas no ambiente.

VII. CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho propôs um algoritmo de escalonamento para um ambiente de nuvem federada, que considera fatores como localidade dos arquivos, capacidade computacional dos recursos, tipo de nuvens que os recursos pertencem, os serviços disponibilizadoo por cada recurso e que realize o balanceamente de carga entre os recursos. Esses objetivos foram alcançados por meio da implementação do AcoSched, utilizando os conceitos da heurística que é baseada na análise de uma colônia de formigas real.

As mudanças na plataforma de nuvem federada modificaram a forma de comunição entre os recursos da federação, que passou de troca de mensagens para Chamada de Processos Remotos(RPC), além da descentralização do controle da nuvem com a utilização de uma ferramenta de coordenação para ambientes de sistemas distribuídos. O RPC, permitiu uma programação simples e de fácil manutenção para a implementação do ZooNimbus.

Para trabalhos futuros, é proposto a substituição da latência utilizada para realizar o escalonamento por outra variável relacionada, ou uma nova proposta na forma de cálculo da latência, já que pode haver problemáticas que atrapalhem o cálculo da latência em uma rede. Outra proposta é a análise do funcionamento do AcoSched ao realizar o escalonamento de tarefas para uma federação de nuvem com um número maior de recursos disponíveis, o que levará a um possível ajuste nos valores de controle e, portanto, a diminuição do tempo total de execução de das tarefas.

REFERÊNCIAS

[1] Langmead B., Trapnell C., Pop M., and Salzberg S. Ultrafast and memory-efficient alignment of short dna sequences to the human genome. In *Genome Biology, 10(3):R25+*, 2009.

[2] Carlos A. L. Borges. Escalonamento de tarefas em uma infraestrutura de computação em nuvem federada para aplicações em bioinformática. http://hdl.handle.net/123456789/377, 2012. Departamento de Ciência da Computação, Universidade de Brasília.

[3] Tusa F. Villari M. Puliafito A. Celesti, A. How to enhance cloud architectures to enable cross-federation. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 337 –345, july 2010.

[4] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In *10th IEEE International Conference on Computer and Information Technology*, 2010.

[5] Yee-Ming Chen and Wen-Chien Wang. An adaptive rescheduling scheme based heuristic algorithm for cloud services applications. In *International Conference on Machine Learning and Cybernetics, Guilin*, 2011.

[6] Tomié D., Muftié O., and Biocié B. A novel scheduling approach of e-learning content on cloud computing infrastructure. In *MIPRO*, 2011.

[7] The Apache Software Foundation. Apache avro^{TM} 1.7.4 documentation. http://avro.apache.org/docs/current/, 2013. Acessado online em 30 de junho de 2013.

[8] The Apache Software Foundation. The apache software foundation. http://www.apache.org/, 2013. Acessado online em 30 de maio de 2013.

[9] The Apache Software Foundation. Apache zookeeper. http://zookeeper.apache.org/, 2013. Acessado online em 30 de maio de 2013.

[10] Franco F. M. de M. Houaiss A., Villar M. de S. *Dicionário Houaiss da língua portuguesa*. São Paulo, 1 edition, 2009.

[11] Guangyu Z. Yushuang D. Wang D. Kun L., Gaochao X. Cloud task scheduling based on load balancing ant colony optimization. In *Chinagrid Conference (ChinaGrid), 2011 Sixth Annual*, pages 3 –9, aug. 2011.

[12] Kun Li, Gaochao Xu, Guangyu Zhao, Yushuang Dong, and Dan Wang. Cloud task scheduling based on load balancing ant colony optimization. In *Sixth Annual China Grid Conference*, 2011.

[13] Amazon Web Services LLC. Amazon elastic compute cloud. http://aws.amazon.com/pt/ec2/, 2012. Acessado online em 12 de Novembro de 2012.

[14] Nawfal A. M., Holmes B., Mamat A., and Shamala K. S. Sharing-aware intercloud scheduler for data-intensive jobs. In *International Conference on Cloud Computing, Technologies, Applications & Management*, 2012.

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

[16] G. de Oliveira. Acosched um algoritmo de escalonamento de tarefas para nuvem federada, 2013. Departamento de Ciência da Computação, Universidade de Brasília.

[17] Sotiriadis S., Bessis N., and Antonopoulos N. Towards inter-cloud schedulers: A survey of meta-scheduling approaches. In *International Conference on P2P, Parallel, Grid, Cloud and Internet Computing.*, 2011.

[18] Borges C. Araujo A. Gallon R. Holanda M. Walter M. E. Togawa R. Setubal J. C. Saldanha H., Ribeiro E. *BioInformatics*. In Tech, 2012.

[19] Hu Wu, Zhuo Tang, and Renfa Li. A priority constrained scheduling strategy of multiple workflows for cloud computing. In *ICACT*, 2012.