Federação de *Clouds* e Atributos de Segurança

Luciano Barreto, Joni da Silva Fraga, Frank Siqueira

Universidade Federal de Santa Catarina
Caixa Postal 476 – 88040-900 – Florianópolis – Santa Catarina – Brasil

{lucianobarreto, fraga}@das.ufsc.br, frank@inf.ufsc.br

Resumo. *Atualmente, o conceito de Federações tem atraído a atenção da comunidade de pesquisa na área de computação em nuvem. Em uma federação de clouds, provedores de nuvem criam relações de confiança e compartilham recursos, seja por escassez momentânea ou com o intuito de oferecer uma maior gama de recursos. Este artigo apresenta uma infraestrutura para formação de uma federação de clouds, que define a forma de organização e estratégias para busca e aquisição de recursos na federação. Além disso, de modo a permitir o gerenciamento de identidades entre domínios, a infraestrutura proposta incorpora mecanismos de autenticação e autorização no contexto da federação. Resultados de simulações mostram a viabilidade da infraestrutura proposta.*

Abstract. *Nowadays, the concept of Federations has drawn attention of the research community in the area of cloud computing. In a cloud federation, cloud providers build trust relationships and share resources, aiming to overcome momentary resource shortage or to offer a wider range of resources to their users. This paper introduces an infrastructure for building a cloud federation, which defines organization principles and strategies for searching and acquiring resources in a federation. In addition, aiming to allow cross-domain identity management, the proposed infrastructure incorporates authentication and authorization mechanisms in the context of a federation. Simulation results show the feasibility of the proposed infrastructure.*

1. Introdução

Nos últimos anos, a computação em nuvem tem sido foco de grande parte dos esforços de pesquisa da comunidade de sistemas distribuídos. Muitas das noções pré-estabelecidas de sistemas e de aplicações distribuídas vêm sendo revistas diante dos requisitos postos por estes novos e complexos ambientes de processamento distribuído.

As grandes demandas que esses sistemas devem suportar têm feito dos mesmos foco de experimentações e de busca de novos conceitos, objetivando manter bons níveis de aproveitamento de recursos com alta qualidade de serviço. O conceito de *federação*, que vem sendo amplamente empregado no sentido de facilitar o gerenciamento de identidades, começa a ser explorado na literatura de *clouds*, visando a cooperação entre diferentes provedores. A integração em federações permite que provedores de *clouds* colaborem de forma a compartilhar recursos no atendimento a demandas de usuários. Os tipos de compartilhamento podem ocorrer em diversos níveis de recursos (máquinas virtuais, armazenamento, redes virtuais, *software*, etc.) e em diferentes situações, tais como: a falta de recursos em um provedor para suportar grandes demandas, o fornecimento de serviços

ou recursos originalmente não oferecidos por um provedor, a distribuição de uma aplicação em diferentes provedores (definida pelo próprio usuário), em caso de colaboração entre pequenos provedores de *clouds*, dentre outros.

O interesse atual na investigação do uso do conceito de federação de *clouds* é, portanto, justificável. No entanto, o agrupamento de provedores de *clouds* formando redes de confiança que atendem um vasto número de usuários coloca também grandes desafios na autenticação e autorização destes usuários junto aos provedores. Para contornar possíveis dificuldades no trato de aspectos de segurança, modelos de gerenciamento de identidades se tornam importantes para esses grandes sistemas.

O foco principal deste trabalho é a apresentação de uma proposta de modelo de federação de *clouds* no qual são identificadas as principais entidades que suportam o conceito de federação. O modelo a ser proposto descreve ainda os protocolos usados na busca e aquisição de recursos na federação de *clouds*, em atendimento às demandas de usuários. Estes protocolos comportam também mecanismos que visam atender requisitos de segurança, como os controles de autenticação e de autorização na federação de *clouds*.

Os serviços de federação são fornecidos a partir de uma infraestrutura de serviços e de um provedor de identidades. As diversas entidades (provedores e usuários) da federação confiam nos serviços dessa infraestrutura e nas autenticações do provedor de identidades, o que permite que provedores de *clouds*, individualmente, não se preocupem em manter extensas bases de dados com identidades e atributos de usuários. Neste texto, além de apresentarmos o modelo proposto e avaliarmos os resultados obtidos através de simulações, confrontamos as características funcionais do modelo com as propostas relacionadas presentes na literatura.

2. Federação de Provedores de *Cloud*

Em [Grozev and Buyya 2012], o conceito de federação de *clouds* é apresentado como um conjunto de provedores que interconectam suas infraestruturas com o objetivo de compartilhar recursos. Porém, além do termo federação, outros termos são usados para nomear a cooperação de provedores no compartilhamento de recursos. Entre outros termos, são citados na literatura: *Inter-cloud* [Grozev and Buyya 2012], *Multi-cloud* [Grozev and Buyya 2012] e *Cross-Cloud* [Celesti et al. 2010]. Embora as diferenças de nomenclatura e mesmo que muitas das propostas para a formação de redes de confiança de *clouds* tenham diferentes focos, o objetivo final é sempre o uso de serviços de diversos provedores de *clouds* para atender a demanda e o *SLA* (*Service-Level Agreement*) de cliente.

2.1 Proposta de Modelo de Federação

O modelo introduzido neste artigo foi desenvolvido para que usuários, diante da necessidade de recursos computacionais, solicitem seus *SLAs* para que uma entidade de suporte, baseada nas informações de demanda, busque por recursos na federação de *clouds* que atendam a requisição do usuário. Com isso, os usuários não se preocupam em como ou onde os recursos estão disponíveis para seu uso. A localização dos recursos é sempre controlada por uma infraestrutura global de serviços que suporta a federação de *clouds*.
Uma visão geral do modelo proposto é ilustrada pela Figura 1, na qual as diferentes entidades do modelo são representadas. A primeira destas corresponde a uma *API* denominada de *Cloud User*, que permite aos usuários interagir com a federação, enviando *SLAs* e requisições de recursos.

As interações do usuário com a federação no modelo proposto são feitas através de uma infraestrutura chamada de Suporte da Federação (*SF*), que é responsável por receber as requisições de recursos e instanciar *brokers* para negociar e controlar diretamente o *SLA* de seu usuário na federação de *clouds*. Para cada *SLA* de usuário, um *broker* é instanciado e permanece ativo até o fim deste contrato. Cada *broker* atua em nome do seu usuário na aquisição dos recursos, entregando *tokens* de autenticação aos provedores de *cloud* de forma a comprovar estes requisitantes como usuários válidos na federação.

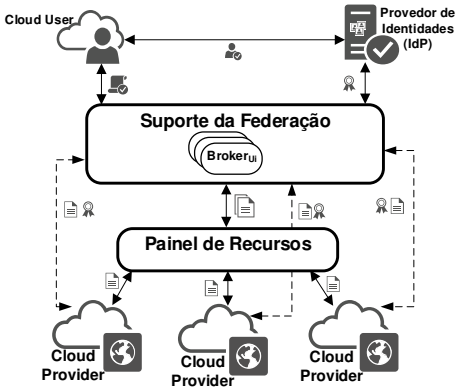

Figura 1 – Visão geral da proposta de federação

O modelo pressupõe o envolvimento de diversos provedores de *cloud* (*Cloud Providers*) oferecendo diferentes tipos de serviço e zelando única e exclusivamente pelos seus próprios recursos. Estes provedores interagem com os *brokers* no momento da busca e aquisição dos recursos. Após este processo, os recursos considerados serão mantidos disponíveis aos usuários envolvidos. Os provedores de *cloud* mantêm seus próprios controles de segurança (autorização), de forma que somente ofereçam recursos para usuários que tenham a posse de um *token* de autenticação.

Nas requisições de recursos, as interações entre *brokers* e provedores de *cloud* no modelo proposto são concretizadas através do componente Painel de Recursos (*PR*), que se comporta de forma semelhante a um serviço de eventos. Para receberem pedidos vindos de *brokers*, provedores devem se registrar no Painel como receptores de notificações adequadas a seus recursos disponíveis. No Painel de Recursos são mantidas listas de provedores ativos e registrados, agrupados pelo tipo de recurso que tornam disponível na federação. No recebimento de demandas, o Painel verifica quais são os recursos solicitados e notifica diretamente os provedores. Os provedores então respondem ao *broker* correspondente informando sua disponibilidade dos recursos solicitados.

Um Provedor de Identidades (*IdP*) foi introduzido no modelo com funções de manter, gerenciar e autenticar usuários e seus atributos na federação. Cada usuário, no acesso ao Suporte da Federação (*SF*), deve inicialmente se autenticar junto a este *IdP*. Autenticidades verificadas geram *tokens* (ou asserções) de autenticação, que devem garantir aos usuários validados a busca e o uso de recursos nos diversos provedores de *cloud* da federação. Os provedores de *cloud* confiam nas autenticações de usuários do *IdP* e recursos só são liberados para usuários previamente autenticados na federação.

2.2 Componentes e Protocolos

As próximas subseções apresentam os algoritmos que descrevem como são feitas as trocas na busca e alocação de recursos entre as entidades do modelo. Estas buscas e acessos

aos recursos são acompanhados por controles de autenticação e de autorização dos usuários da federação.

2.2.1 *Cloud User*

O Algoritmo 1 apresenta o processo de solicitação de recursos na federação, onde o usuário inicialmente gera seu *SLA* (linhas 13 – 17), especificando os parâmetros de contrato (preço, *uptime* mínimo, qualidade mínima de serviço, dentre outros). Os tipos de cada recurso desejado são selecionados (linhas 19 – 22, Algoritmo 1) na sequência. A requisição $fedReq_i$ é montada (linha 25), armazenada (linha 26) e por fim remetida ao *SF* (linhas 27). Uma vez enviada a requisição, em um primeiro momento o usuário deve receber um pedido de autenticação ($authCReq_i$) do *IdP*. O usuário então responde ao *IdP* com suas credenciais (usuário/senha, certificado, etc.) através da mensagem $authCred_i$ (linhas 28–30). Após esse passo, o usuário aguarda pelos ponteiros (endereços) dos recursos, os quais serão enviados pelo *SF*, por meio do *broker* alocado para o seu *SLA*, em uma mensagem do tipo $fedRep_i$ (linhas 31 e 32).

Algoritmo 1 – *Cloud User*

Algoritmo 1. *Cloud User* U_i

```
Structures
1:    Resource {
2:      Type;
3:      Amount;}
4:  Type:
5:    VmTypeA = {1ghz cpu, 1gb mem, ...}
6:    VmTypeB = {2 ghz cpu, 2 gb mem, ...}
7:  Init:
8:    n_u ← 0;
9:    fedReqList[] ← Ø;
10:   res[] ← Ø;

11:  upon ResourceRequest do
12:    //Create SLA requirements. Example:
13:    price ← ≥ X;
14:    uptime ← ≤ 99%;
15:    QoS ← ≥Y;
16:    ....
17:    sla_i ← <SLA, price, uptime, QoS, ...>
18:    //Create resource requirements. Example:
19:    res ← res ∪ Resource(VmTypeA, 2);
20:    res ← res ∪ Resource(VmTypeB, 3);
21:    …
22:    resReq_i ← <RESREQ, res>;
23:    n_u++;
24:    id_req ← <IDREQ, id_Ui, n_u>;
25:    fedReq_i ← <FEDREQ, id_Ui, id_req, sla_i, resReq_i, sign_ui, cert_ui>
26:    fedReqList ← fedReqList ∪ fedReq_i;
27:    send fedReq_i to SF

28:  upon Receive authCReq_i from IdP do
29:    authCred_i ←<AUTHCRED_Ui, id_Ui, id_req, credentials, sign_ui, cert_ui>;
30:    send authCred_i to IdP

31:  upon Receive fedRep_i from B_i do
32:      resourceRequested ← fedRep_i.fedAlRes;
```

Todas as mensagens trocadas entre as entidades são assinadas de modo a garantir sua a autenticidade e integridade. Ou seja, as entidades do modelo só trocam informações com outras entidades credenciadas e devidamente validadas. Por exemplo, com a mensagem $fedReq_i$ (linha 25) é enviada a sua assinatura $sign_{ui}$ e o certificado de chave pública do assinante ($cert_{ui}$) para validação posterior da mensagem.

2.2.2 Suporte da Federação

A entidade identificada como Suporte da Federação (*SF*) é responsável por receber requisições e instanciar *brokers* para atender as solicitações de usuários. O algoritmo 2 descreve seu comportamento nas interações com o *Cloud User*. Neste algoritmo, ao receber uma requisição $fedReq_i$ (linha 3), o *SF* armazena e recupera algumas informações importantes, como por exemplo, o identificador do usuário (linhas 4 – 6, Algoritmo 2).

De posse do identificador id_{ui} do usuário, o *SF* executa um procedimento de descoberta [Miller 2006] (linha 7) que visa encontrar o *IdP* que armazena as credencias do usuário. Executado esse protocolo de descoberta, o Suporte da Federação (*SF*) envia então uma requisição de autenticação ($authReq_i$) ao *IdP* correspondente (linhas 8 e 9). Após esse passo de autenticação, o *SF* aguarda um *token* de autenticação ($authToken_i$) vindo do *IdP* que confirme o emissor da requisição como usuário válido. Uma vez recebido esse

token, o *SF* faz as verificações necessárias da assinatura do *token* que, caso sua validade seja confirmada, instancia um *broker* para atender a requisição do usuário (linhas 10 -14).

Algoritmo 2 – Suporte da Federação

Algoritmo 2. *Suporte da Federação F*

1. **Init:**
2. $fedReqList \leftarrow \emptyset$;
3. **upon** ***Receive*** $fedReq_i$ **from** U_i **do**
4. $fedReqList \leftarrow fedReqList \cup fedReq_i$;
5. $id_{req} \leftarrow fedReq_i.id_{req}$;
6. $id_{ui} \leftarrow fedReq_i.id_{Ui}$;
7. $IdP \leftarrow$ ***DiscoverIDP***(id_{ui});
8. $authReq_i \leftarrow <AUTHREQ, id_{ui}, id_{req}, sign_F, cert_F>$;
9. ***send*** $authReqi$ ***to*** IdP;
10. **upon** ***Receive*** $authToken_i$ **from** IdP
11. $token_{ui} \leftarrow authToken_{Ui}.token_{ui}$;
12. $id_{req} \leftarrow authToken_i.id_{req}$;
13. **if** ***verifyToken***$(token_{ui}) \wedge fedReqList.get(id_{req})$ **then**
14. $broker_{Ui} \leftarrow$ ***createBroker***$(fedReq_i, token_{ui})$;
15. **end if;**

2.2.3 *Broker*

O *broker* é o agente instanciado com o objetivo de efetuar a negociação e alocação de recursos na federação. O algoritmo 3 descreve o comportamento do *broker* na busca e gerência de um *SLA* de usuário. Para a instanciação de um *broker* (linha 8 - Algoritmo 3), o *SF* passa como parâmetro a requisição do usuário assim como o *token* gerado pelo *IdP* no processo de autenticação. Esse *token* será usado posteriormente nos controles de autorização quando da requisição de recursos em provedores de *cloud* da federação.

O próximo passo desse *broker* recém-instanciado é então encaminhar a requisição $fedReq_iB_i$, em nome de seu usuário, ao Painel de Recursos (*PR*) (linha 14). Estas mensagens incorporam o identificador do *broker* e a requisição do usuário. Vale ressaltar que a assinatura desta mensagem é feita com a chave privada do Suporte da Federação (*SF*) e é o seu certificado que vai com esta requisição de modo a validar esta assinatura. A requisição $fedReq_iB_i$, com as solicitações do usuário, é então enviada pelo *broker* ao *PR* (linha 15). Como consequência, um temporizador é iniciado pelo *broker* para controlar em prazo estipulado as recepções de respostas a esta requisição (linha 16). No caso de decorrência deste prazo (linhas 44 – 47), a mensagem $fedReq_iB_i$ é enviada novamente ao *PR* e um novo prazo é estimado para o recebimento das respostas correspondentes.

No recebimento destas mensagens de resposta ($fedRes_i$), o *broker* verifica se o provedor correspondente pode atender totalmente os recursos solicitados na requisição do usuário (linhas 17-19). Em caso afirmativo, o temporizador iniciado para espera de respostas é removido e uma resposta $fedReq_iAck_i$ é enviada a este provedor (linhas 21-23), informando a concordância em alocar todos os recursos indicados. Caso a resposta do provedor não indique o atendimento total da requisição do usuário, o *broker* então armazena essa resposta para uma possível alocação distribuída entre os diversos provedores de *cloud* da federação (linhas 24 - 26). Quando a soma dos recursos que podem ser suportados parcialmente pelos provedores de *cloud* for suficiente para o atendimento das necessidades do usuário (linha 27), o *broker* então remove o temporizador (linha 28) e ordena a lista de provedores que atenderam a solicitação de recursos via Painel (linha 29). Esta ordenação usa o critério de número máximo de recursos livres por provedor de *cloud*, tentando com isto minimizar o número de provedores que atendam a requisição do usuário.

Efetuado o envio da confirmação para que os recursos sejam alocados, o *broker* então fica no aguardo de mensagens $fedAllocAck_i$. Nessas mensagens são enviados os ponteiros que indicam onde os recursos estão disponíveis. Com o recebimento dessas mensagens $fedAllocAck_i$ (linha 37 - Algoritmo 3), o *broker* armazena os ponteiros em uma

lista e verifica se o número de ponteiros recebidos atende ao número de recursos da requisição do cliente (linha 40). Caso a verificação seja verdadeira, uma mensagem $fedRep_i$ é gerada com os ponteiros dos recursos (linha 41) e enviada para o *Cloud User* (linha 42, Algoritmo 3), finalizando o processo de aquisição de recursos.

Algoritmo 3 - *Broker*

Algoritmo 3. *Cloud Broker* B_i

```
1:  Init:
2:  resReq ← Ø;
3:  partFedRes ← Ø;
4:  authTkn_Ui ← Ø;
5:  fedAlRes ← Ø;
6:  timeout ← initialValue;
7:  δ_B ← Ø
8:  upon CreateBroker(fedReq_i, token_Ui) do
9:    idU ← fedReq_i.id_req.id_ui;
10:   id_Bi ← <IDBI, B_i, idU>;
11:   id_req ← fedReq_i.id_req;
12:   resReq ← fedReq_i.resReq_i.res;
13:   authTkn_Ui ← tokenUi;
14:   fedReq_iB_i ← <FEDRB, id_Bi, id_req, fedReq_i, sign_F, cert_F>;
15:   send fedReq_iB_i to RP;
16:   δ_B.time ← time();
17: upon Receive fedRes_i from C_j do
18:   id_req ← fedRes_i.id_req;
19:   if fedRes_i.canff = "Full" then
20:     nRes ← "Full";
21:     δ_B.time ← ⊥;
22:     fedReq_iAck_i ← <FEDREQACK, id_Bi id_req, fedRes_i, ↩
            Full, sign_F, cert_F>;
23:     send fedReq_iAck_i to C_j;
24:   else
25:     partFedRes ← partFedRes ∪ fedRes_i;
26:     fedResCount ← fedResCount + fedRes_i.MaxFreeRes;
27:     if (fedResCount ≥ | resReq|) then
28:       δ_B.time ← ⊥;
29:       orderByMaxFreeResources(partFedRes);
30:       for all C_j ∈ partFedRes
31:         nRes = [i…n];
32:         fedReq_iAck_i ← <FEDREQACK, id_Bi, id_req, ↩
              fedRes_i, nRes, authTkn_Ui, sign_F, cert_F>;
33:         send fedReq_iAck_i to C_j;
34:       end for;
35:     end if;
36: end if;
37: upon Receive fedAllocAck_i do
38:   id_req ← fedAllocAck_i.id_req;
39:   fedAlRes ← fedAlRes ∪ fedAllocAck_j.resPointers;
40:   if (|fedAlRes| ≥ |resReq|) then
41:     fedRep_i ← <FEDREPLY, id_Bi,id_req, ↩
            fedAllocRes, sign_F, cert_F>;
42:     send fedRep_i to U_i;
43:   end if;
44: upon (time() - δ_B.time) ≥ timeout do
45:   send fedReq_iB_i to RP;
46:   δ_B.time ← time();
47:   timeout ← NewEstimation(timeout);
```

2.2.4 Painel de Recursos

Como citado anteriormente, um serviço de eventos especializado, nomeado Painel de Recursos (*PR*), integra o modelo de federação proposto, permitindo com isso que os provedores de *cloud* sejam notificados sobre demandas de recursos. O comportamento do *PR* é explicitado no Algoritmo 4. Ao receber uma requisição $fedReq_iB_i$ (linha 8, Algoritmo 4), esse serviço (*PR*) deve notificar aos provedores apropriados sobre a demanda correspondente. O *PR* percorre então as listas de provedores (linhas 12 - 23), enviando mensagens $reqNotif_i$ para todos os provedores que oferecem determinados tipos de recursos.

Diante da disponibilidade ou indisponibilidade momentânea de recursos, os provedores de *cloud* da federação podem se inscrever ou pedir para ser removidos das listas associadas a tipos de recursos. Ou seja, essas listas evoluem refletindo a dinamicidade da federação em termos da disponibilidade de recursos. O *PR* oferece duas operações em sua interface que permitem que provedores entrem ou saiam das listas associadas: $memEnter_m$ e $memExit_m$. O *PR*, ao receber uma mensagem $memEnter_m$, adiciona o provedor em listas (associadas) de tipos de recursos que este provedor dispõe, deixado o mesmo habilitado para receber notificações correspondentes (linhas 25 - 32). No recebimento de uma mensagem $memExit_m$, o *PR* remove o provedor das listas associadas de tipos de recursos que o mesmo já não dispõe (linhas 34 – 40).

2.2.5 *Cloud Provider*

Provedores de *cloud* são os fornecedores de recursos na federação. Uma vez estabelecidos os contratos (via *brokers*), os usuários passam a ter acesso direto aos recursos alocados, segundo os *SLAs* estabelecidos.

Algoritmo 4 – Painel de Recursos

Algoritmo 4. *ResourcePanel RP*

```
1.  Structures
2.    Member{
3.      Type} // VmTypeA, VmTypeB
4.      Name[]}; //C1, C2, C3....
5.  Init:
6.  memList ← Members;
7.  reqList ← Ø;

8.    upon Receive fedReqiBi from Bi
9.      idreq ← fedReqiBi.idreq;
10.     idRP ← <IDRP, idRP, idReqP>;
11.     reqList ← reqList ∪ fedReqiBi;
12.     for all Type ∈ fedReqiBi.resReqi.res.Type do
13.       case Type is
14.         VmTypeA:
15.             type ← VMTypeA;
16.             reqNotifi ← <REQNOTIF, idRP, idreq, fedReqiF, ↩
                    type, signrp, certrp>;
17.             send reqNotifi to all Cj ∈ MemList.Type(VmTypeA);
18.         VmTypeB:
19.             type ← VMTypeB;
20.             reqNotifi ← <REQNOTIF, idRP, idreq, fedReqiF, ↩
                    type, signrp, certrp>;
21.             send reqNotifi to all Cj ∈ memList.Type(VmTypeB);
22.       ...
23.     end case
24.   end for

25.   upon Receive memEnterm from Cj
26.     case memEnterj.Type is
27.       VmTypeA:
28.         memList.VmTypeA ←memList.VmTypeA ∪ memEnterm.Cj;
29.       VmTypeB:
30.         memList.VmTypeB ←memList.VmTypeA ∪ memEnterm.Cj;
31.       ...
32.     end case

33.   upon Receive memExitm from Cj
34.     case memExitj.Type is
35.       VmTypeA:
36.         memList.VmTypeA ←memList.VmTypeA - memEnterm.Cj;
37.       VmTypeB:
38.         memList.VmTypeB ←memList.VmTypeB - memEnterm.Cj;
39.       ....
40.     end case
```

Ao receber uma notificação (*reqNotif*$_i$) do Painel de Recursos, o provedor de *cloud* primeiramente recupera o contrato de *SLA* da requisição do usuário e verifica se o mesmo está de acordo com os parâmetros e requisitos definidos para o fornecimento de recursos (linhas 9-14, Algoritmo 5). Caso o provedor possa atender ao *SLA*, a disponibilidade dos recursos solicitados é verificada (linha 15).

O atendimento pelo provedor de uma solicitação de recursos pode ocorrer de duas maneiras: de forma total, onde o provedor pode atender toda a demanda do usuário, ou parcial, quando somente parte dos recursos requisitados estão disponíveis. Se a quantidade de recursos no provedor é suficiente para atender por completo a solicitação (linha 16), o provedor de *cloud* faz então a reserva desses recursos (linha 17) e monta a mensagem *fedRes*$_i$ que informa o atendimento integral (*canff* ← *Full*) pelo provedor da solicitação do usuário. Essa mensagem de resposta à notificação *reqNotif*$_i$ é enviada diretamente ao *broker* solicitante (linhas 19-20). Um temporizador é iniciado após o envio da mensagem *fedRes*$_i$ (linha 21) para controlar um prazo definido segundo políticas do provedor de *cloud*. Este prazo é usado para liberar os recursos reservados, caso não haja resposta de confirmação do *broker* dentro do limite de tempo estipulado (linhas 45 - 46).

No caso da impossibilidade de atendimento total da requisição do usuário, o provedor monta sua mensagem *fedRes*$_i$, informando o atendimento parcial da solicitação (*canff* ← *Partial*) e também a quantidade disponível (*maxFreeRes*) de recursos (linha 19). Nesse último caso, os recursos são também reservados e um temporizador é iniciado para que os recursos sejam liberados quando não houver resposta a essa reserva (linha 21). No recebimento de uma mensagem *fedReqiAck*$_i$ (linha 31), o provedor de *cloud* recupera e valida o *token* de autenticação do usuário (linhas 33-34). Esta validação envolve a verificação dos dados de assinatura do *token* (data de emissão e duração do mesmo), dentre outros atributos do usuário contidos neste *token*, e corresponde ao controle de autorização de um provedor de *cloud*. A apresentação do *token* corresponde à posse de uma competência (ou *capability*) na nomenclatura de controle de acesso [Landwehr 2001].

Algoritmo 5 – *Cloud Provider*

Algoritmo 5. *Cloud C_j*

```
1.  Init:
2.  resReq ← Ø;
3.  localRes ← Ø;
4.  resPointers ← Ø;
5.  fedReqAck ←Ø;
6.  timeout ← initialValue;
7.  available ← True;
8.  δi ← Ø;

9.  upon Receive reqNotifi from RP do
10.   slau ← reqNotifi.fedReqiF.fedReqi.slai;
11.   type ← reqNotifi.type;
12.   resReq ← reqNotifi.fedReqiBi.fedReqi.res(type);
13.   idreq ← reqNotifi.idReq;
14.   if agreeSLA(slau) then
15.     localRes ← getFreeResources(resReq, type);
16.     if |localRes| ≥ | resReq| then
17.       reserveRes(idreq, localRes);
18.       canff ← Full;
19.       fedResi ← <FEDRES, idCj, idreq, canff, signcj, certcj>;
20.       send fedResi to Bi;
21.       δ.time ← time();
22.     else
23.       canff ← Partial;
24.       reserveRes(reqID, localRes);
25.       maxFreeRes ← |localRes|;
26.       fedResi ← <FEDRES, idCj, idreq, canff, ↩
              maxFreeRes, signcj, certcj>;
27.       send fedResi to Bi;
28.       δ.time ← time();
29.     end if
30.   end if

31. upon Receive fedReqiAcki from Bi
32.   idreq ← fedReqiAcki.idreq;
33.   tokenui ← fedReqiAck.authTknUi;
34.   if validadeToken(tokenui) then
35.     δ.time ← ⊥;
36.     idBi ← fedReqiAcki.idBi;
37.     if (resReq = "Full") then
38.       resPointers ← Allocate(idreq);
39.     else
40.       resPointers ← Allocate(idreq [i…n]);
41.       fedAlAcki ←<FEDALCK, idCj, idBi, idreq, ↩
          resPointers, signcj, certcj>;
42.       send fedAlAckj to Bi;
43.     end if;
44.   end if;

45.   upon (time() - δ.time) ≥ timeout do
46.     freeResource(idreq);

47. upon LocaResource.Type.count ≤ 0 do
48. if available = True then
49.   memExitm ← <MEMEXIT, Cj, Type>
50.   available ← False;
51.   send memExitm to RP
52. end if

53. upon LocaResource.Type.count > 0 do
54. if available = False then
55.   memEnterm ← <MEMENTER, Cj, Type>
56.   available ← True;
57.   send memEnterm to RP;
58. end if;
```

Com a validação do *token*, o provedor remove o temporizador que controla a reserva e os recursos referentes à requisição são então alocados (linhas 35 - 40). É importante ressaltar que, no caso de atendimento parcial, o provedor de *cloud* não necessariamente aloca todos os recursos previamente reservados (*maxFreeRes,* na linha 26). O *broker*, no envio de sua mensagem $fedReq_iAck_i$, define a quantidade de recursos que deseja no atendimento parcial de cada provedor selecionado. Após alocar os recursos, o provedor de *cloud* responde ao *broker*, através da mensagem $fedAlAck_j$, indicando os ponteiros dos recursos alocados (linhas 41-42).

Os provedores devem também gerenciar a evolução dinâmica da disponibilidade de recursos. Com o decorrer do tempo, tipos de recursos podem ficar indisponíveis ou disponíveis nestes provedores. Diante destas variações, os provedores podem remover suas inscrições no *PR* para receberem notificações sobre determinados tipos de recursos, assim como podem se inscrever em listas associadas quando os recursos correspondentes estiverem novamente disponíveis. Para essas evoluções de seus registros em listas associadas de tipos de recursos, o provedor dispõe dos envios das mensagens $memEnter_m$ e $memExit_m$ para ser adicionado ou removido de listas associadas no *PR* (linhas 47 - 58).

2.2.6 Provedor de Identidades

A autenticação de usuários é importante para o acesso ao Suporte da Federação, bem como para a alocação de recursos junto aos provedores de *Cloud*. Diante disso, o modelo proposto adota um gerenciamento de identidades centralizado [Jøsang et al. 2005], onde um provedor de identidades autentica os usuários dos serviços para o *SF* e provedores de *cloud*. O algoritmo 6 mostra as interações do *IdP* da federação no modelo proposto.

Ao fazer o seu primeiro acesso ao *SF*, o usuário é direcionado ao *IdP* para se autenticar. Para isso, o *SF* envia uma requisição de autenticação ($authReq_i$) ao *IdP* da federação (linhas 8 e 9, Algoritmo 2). O *IdP*, ao receber essa mensagem (linha 3, Algoritmo 6), verifica se o usuário não está autenticado, através da existência do *token* de autenticação correspondente em sua lista de *tokens* (linha 7, Algoritmo 6). Caso o usuário já tenha sido autenticado, o *IdP* responde ao *SF* enviando o *token* recuperado (linhas 8). Caso o usuário ainda não tenha se autenticado, o *IdP* requisita então que este informe suas credencias, enviando uma mensagem $authCReq_i$ ao *Cloud User* (linhas 10 e 11). No recebimento das credenciais do usuário, através da mensagem $authCred_i$, uma vez validadas estas credencias (linhas 12-14) o *IdP* gera um *token* de autenticação devidamente assinado ($authToken_i$), o armazena e, por fim, o envia para o *SF* que solicitou a autenticação do usuário (linhas 15 - 19, Algoritmo 6).

Algoritmo 6 – Provedor de Identidades

Algoritmo 6. *Identity Provider IdP*

```
1.  Init:
2.  tokenList[];
3.  upon Receive authReq_i from F
4.    U_i ← authReq_idp.id_Ui;
5.    id_req ← authReq_idp.id_req;
6.    authToken_i ← tokenList.get(U_i);
7.    If authToken_i ≠ null then
8.      send authToken_i to F;
9.    else
10.     authCReq_i ← <AUTHCREQ, id_IdP, id_req, Sign_IdP, Cert_IdP>;
11.     send authCReq_i to U_i;
12.   upon Receive authCred_i from U_i
13.     credentials ← authCred_ui.credentials;
14.     if verifyCredentials(credentials) then
15.       U_i ← authCred_i.id_Ui;
16.       id_req ← authCred_i.id_req;
17.       authToken_i ← <AUTHTOKEN, id_Ui, id_req, signature, ↩
                  RevokeDate, sign_idp, cert_idp>;
18.     tokenList ← tokenList ∪ authToken_i;
19.     send authToken_i to F;
20.   end if;
```

2.2.7 Considerações sobre os Algoritmos

Algumas considerações devem ser feitas sobre as operações efetuadas pelo *broker* (Algoritmo 3). Na abordagem proposta, no caso de somente um provedor atender por completo a requisição do usuário, não é verificado se este provedor é o que melhor atende a requisição (em termos de *QoS*, ou outro fator). Simplesmente o primeiro provedor de *cloud* que atende a requisição completamente é o escolhido (linhas 17-23, algoritmo 3). Um modelo que aguardasse as diversas respostas e as analisasse, usando critérios de *QoS* ou qualquer outro, poderia gerar um melhor benefício para o usuário da federação. Porém, nesse momento, este não foi o foco do trabalho. Além disso, na verificação de *SLAs* de usuário, não entramos em discussões sobre negociações dos parâmetros descritos no contrato. Na abordagem proposta o provedor de *cloud* aceita ou não o *SLA* do usuário (linha 14, Algoritmo 5). Porém um protocolo que levasse em conta uma negociação entre o que pode ser oferecido pelo provedor e as necessidades de usuário poderia tornar esta relação de uso de recursos mais flexível.

A segunda consideração a ser feita é sobre o atendimento da requisição do usuário por diversos provedores de *cloud* (linhas 24 - 33, algoritmo 3). Nesse caso, por simplicidade, foi adotada uma heurística que ordena a lista de provedores que devem atender a requisição do usuário, priorizando aqueles com maior número de recursos disponíveis. Podem ocorrer situações onde são excluídos provedores que podem atender totalmente a requisição. É o caso quando a soma dos recursos parciais é atingida antes do recebimento de uma resposta com a possibilidade citada. Assim, a primeira consideração sobre o *broker*, ou seja, a aplicação de uma heurística mais complexa que analisasse os diversos parâmetros das respostas parciais, poderia trazer benefícios. Outro ponto a ser considerado é a reserva de recursos no provedor de *cloud* (linhas 17 - 28, algoritmo 5). No momento da reserva, o provedor deixa de oferecer esses recursos a outras demandas,

o que não é interessante para os provedores. Nesse sentido, a definição do parâmetro $\delta_i.time$ (linha 28, algoritmo 5) deve levar em conta um *tradeoff* entre poder contribuir com o atendimento aos contratos e não perder outras demandas.

3. Simulações e Análises

O objetivo central que colocamos nas simulações realizadas é obter medidas aproximadas dos custos em mensagens e do tempo necessário para alocar recursos na federação. Para tanto, um modelo de simulação foi implementado. O modelo criado teve como base o simulador *CloudSim* [Calheiros et al. 2011], no qual cada entidade do modelo proposto foi mapeada como uma *CloudSim Entity*, representando os diferentes atores da federação, e as trocas de mensagens foram mapeadas como *CloudSim Events*.

3.1 Definição dos parâmetros do modelo

Vários parâmetros tiveram que ser definidos para que fosse realizada a simulação. Para cada um dos eventos trocados entre as entidades foi atribuído um valor de *delay*, representando os tempos nas trocas de mensagens. A obtenção destes valores de *delay* tomou como base experimentos realizados com a plataforma de *cloud Azure* da Microsoft. Várias medições de largura de banda e latência entre servidores de diferentes regiões geográficas foram realizadas. Utilizamos instâncias de servidores das regiões: América do Sul (BRZ), Europa (EUR), Estados Unidos (USA) e Sudeste da Ásia (ASI).

Outro parâmetro usado em nossas simulações é a quantidade de recursos (*VMs*) requeridos pelos usuários, que denominamos de *workload*. Para tanto, duas abordagens foram definidas: a primeira com o *workload* crescendo de forma linear e a outra com o *workload* na forma de uma distribuição aleatória. O *workload* com crescimento linear é constituído de provedores com um número fixo de recursos livres. Esse *workload* apresenta um aumento gradativo de pedido de recursos por teste, caracterizando um crescimento linear das demandas e definindo também o aumento de provedores atuando sobre as requisições de usuários. No outro modelo de *workload*, os recursos livres, embora fixos no sistema, são distribuídos de forma aleatória entre os diversos provedores de *cloud* da federação. O atendimento das requisições de usuários vai depender da instância de teste. Uma demanda pode ser resolvida com poucos provedores de *cloud*, enquanto em outra instância essa mesma demanda pode envolver um número muito maior de provedores.

Um parâmetro também importante para a avaliar o tempo de resposta aos usuários é o tempo aproximado para a alocação destes dos recursos em um provedor de *cloud*. Usamos novamente a plataforma *Azure* para determinação do tempo para alocar *VMs* do tipo mais básico, e as medições realizadas apontaram para um valor de cerca de 2 minutos. Devido à diferença na ordem de grandeza desse tempo se comparado aos tempos das trocas de mensagens (milissegundos), não incluímos este valor nos gráficos dos resultados. Desta forma, os gráficos dos resultados apresentam os tempos resultantes das trocas do protocolo sem o tempo de alocação dos recursos.

Os algoritmos do modelo proposto apresentam trocas de autenticação. Porém, devido à dificuldade de representar a interação entre usuário e *IdP* no processo de *login*, o modelo simulado considera que o usuário já efetuou o seu *login* na federação. Os pedidos de autenticação enviados ao *IdP* pelo *SF* ($authReq_i$ nas linhas 8 e 9 do Algoritmo 2) fazem parte das simulações. A resposta a esses pedidos é sempre o envio pelo *IdP* do *token* de autenticação ($authToken_i$, na linha 10 do algoritmo 2), considerando sempre que o usuário

já foi previamente autenticado. Por fim, definimos diferentes perfis de simulações que descrevem o posicionamento (geográfico) de cada uma das entidades da simulação. A Tabela 1 apresenta alguns dos perfis de simulação avaliados.

Tabela 1 – Perfis de Simulações

Perf.	User	SF	PR	*IdP*	CP1	CP 2	CP 3	CP 4	CP 5	CP 6	CP 7	CP 8	CP 9	CP 10
1	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ	BRZ
2	BRZ	USA	EUR	BRZ	BRZ	USA	EUR	ASI	USA	EUR	BRZ	ASI	BRZ	BRZ
3	USA	ERU	ASI	USA	BRZ	USA	EUR	ASI	BRZ	USA	EUR	ASI	BRZ	USA

3.2 Modelo de *Workload* Linear

Neste cenário, o parâmetro *workload* foi definido em forma crescente com a inclusão de um provedor a cada instância de teste, compondo uma faixa de recursos que vai de 10 até 100 *VMs*, assumindo que cada provedor fornece o valor fixo de 10 *VMs*. A Figura 2(A) apresenta resultados de tempo de resposta para o usuário obter recursos na federação (recebendo os ponteiros de recursos) em cada um dos *workloads* lineares executados nos diferentes perfis. Já a Figura 2(B) apresenta o crescimento do número de mensagens quando do aumento na alocação de recursos através da inclusão de novos provedores no atendimento dos *workloads*. É possível perceber que o número de mensagens cresce de forma linear. Quando um novo provedor é adicionado para atender a um *workload*, seis (6) novas mensagens são adicionadas aos custos de alocação.

Para prover uma visão dos resultados obtidos no workload linear, tomou-se como exemplo um usuário solicitando 100 recursos na federação e considerando o perfil 3, que dá uma maior diversidade de distribuição geográfica do modelo. Nessa situação o tempo para alocação de recursos na federação ficou em aproximadamente em 2 minutos acrescidos de 351 *ms* consumidos em trocas de comunicações. Ou seja, se tomarmos como referência os 2 minutos que são consumidos na plataforma Azure para alocar VMs, podemos dizer que os custos de alocação em uma federação atendendo um workload linear são praticamente os mesmos e que os custos de comunicação não têm um peso significativo nessas alocações. É claro que consideramos que a disponibilidade de provedores em uma federação torna as respostas imediatas a demandas de usuários.

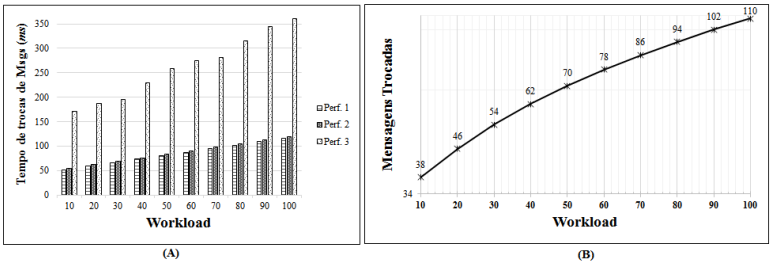

Figura 2 – Resultados do *Workload* Linear

3.3 Modelo com Workload Aleatório

Nestes testes, o *workload* do usuário foi baseado em uma distribuição normal com média em 100 e desvio padrão de 20 nos recursos solicitados. A escolha desses valores não se baseou em nenhuma observação específica, mas sim na ideia de que o número de recursos requisitados pelos usuários estaria entre 80 e 120 *VMs*, o que para grandes usuários de *cloud* pode ser considerado como valor razoável. Os recursos livres por provedor são definidos entre 5 e 150 *VMs*. A definição desses valores para os parâmetros citados foi baseada nos inúmeros testes realizados previamente. Foi observado que escolhas de faixas

menores resultavam em *workloads* sem solução (insuficiência de recursos), o que não é interessante para avaliação do modelo.

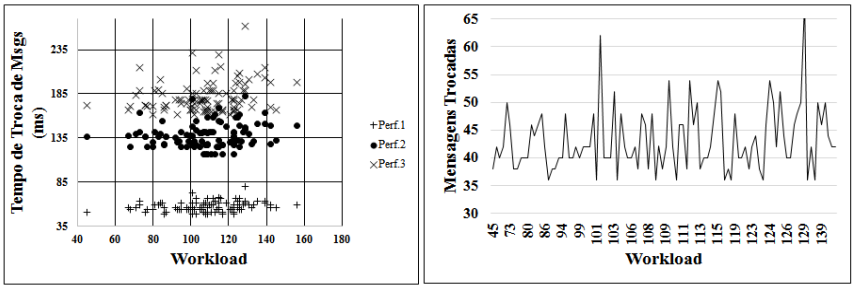

Figura 3 - Resultados do *Workload* Aleatório

As Figuras 3 (A) e (B) apresentam os resultados de tempo de trocas e do número de mensagens trocadas, considerando *workloads* aleatórios. As trocas em cada simulação pode apresentar resultados distintos e, ainda assim, serem considerados como válidos, pois uma execução otimista pode envolver o menor número possível de provedores, enquanto uma execução pessimista pode envolver o máximo de provedores disponíveis.

Tomando a mesma situação da seção anterior – ou seja, um usuário que solicita 100 recursos na federação com Perfil 3 – e considerando o *workload* aleatório, o tempo de alocação de recursos também ficou próximo de 2 minutos, mas foi acrescido de 167 *ms* devido às trocas de mensagens. É possível perceber que a simulação com *workload* aleatório teve tempo de resposta levemente menor do que na carga linear. Essa diferença se dá pelo fato de que, na simulação linear, para a requisição de 100 recursos, obrigatoriamente o suporte irá se utilizar de todos os provedores da federação. Com *workload* aleatório, a aquisição de 100 recursos se dá (na média) com um número menor de provedores.

4. Literatura Relacionada

Um dos primeiros trabalhos para federações de *clouds* foi apresentado por *Buyya* [Buyya et al. 2010], onde uma federação de *clouds* é tida como um modelo de múltiplos provedores interagindo na busca de recursos. Essa busca é em nível provedor-provedor, não havendo um suporte global para tal. Os usuários se associam a um provedor, que fica responsável por atender toda a demanda do cliente. Abordagens similares a essa são também encontradas em [Rochwerger et al. 2009],[Celesti et al. 2010].

Outras experiências de federação, como as apresentadas em [Carlini et al. 2012; Coppola et al. 2012; Villegas et al. 2012], são próximas da proposta neste artigo e fazem o uso de um suporte global na busca recursos. Estes modelos, com uma entidade central controlando buscas e alocações, fazem com que usuários de *clouds* tenham também que interagir com o suporte global que caracteriza a federação. Porém, apesar de não serem claramente discutidos, os registros de usuários podem estar ligados a domínios locais de seus provedores de *cloud* ou controlados a partir deste suporte global da federação. Na verdade, não existe clareza sobre como se dá o gerenciamento de identidades na maioria das abordagens citadas. Sendo sempre passível de interpretação se cada provedor de *cloud*, ou mesmo um suporte global, é responsável pelo seu domínio de usuários. Outra incógnita é como são executados os controles de autenticação e autorização. Estas propostas, em sua maioria, são fundamentadas em modelo tradicional de gerenciamento de

identidades, gerando certas implicações quando federações de *clouds* são definidas a partir de controles locais de identidades. Com domínios de nomes locais e o provimento de recursos se dando no nível provedor-provedor, fica difícil tarifar o usuário pelo seu uso de recursos quando a relação entre provedores pode envolver múltiplos usuários cujos identificadores só tem sentido local.

Uma exceção é a abordagem descrita em [Celesti et al. 2010], onde os autores propõem uma federação de *clouds* baseada em relações provedor a provedor. Essa proposta considera o gerenciamento de identidades federadas. Portanto, as autenticações dos usuários em suas *clouds* locais podem ser transpostas de forma a utilizar recursos de outras *clouds* na federação, sem a necessidade da reapresentação de credenciais. Porém, os custos de manutenção de base de dados com atributos de usuários são locais.

Outra distinção entre as abordagens da literatura é quanto à busca de recursos na federação de *clouds*. Duas abordagens podem ser identificadas: a primeira fazendo uso de um repositório central e a outra baseada em buscas P2P. Nas buscas através de um repositório central [Buyya et al. 2010], os provedores de *cloud* devem periodicamente informar quais são seus recursos disponíveis para a federação a partir deste mecanismo centralizador. Sempre que precisam de recursos, usuários devem recorrer ao repositório central. Apesar de facilitar a busca de recursos, essa técnica pode apresentar fragilidades: o repositório nem sempre apresenta uma visão consistente dos recursos disponíveis nos provedores do sistema distribuído de larga escala que caracteriza uma federação. Isto é consequência das características desses grandes sistemas, normalmente dispersos em complexas distribuições geográficas e sujeitos a grandes demandas.

Nos modelos que fazem uso e trocas P2P na busca de recursos [Celesti et al. 2010; Coppola et al. 2012] a ideia é que, sempre quando necessário, provedores de *cloud* trocam mensagens no sentido de verificar os recursos disponíveis na federação. Essa abordagem resolve o problema de consistência das informações existente na abordagem de repositório central, porém os custos de buscas P2P são geralmente altos em número de mensagens. Tais trocas de mensagens não garantem que os recursos estarão ainda disponíveis quando estes forem definitivamente solicitados para a alocação.

O modelo descrito no presente artigo propõe a adoção de uma estrutura diferente, baseada em um Painel de Recursos. Através do *PR*, um *broker* apresenta uma requisição em nome de seu usuário e, provedores de *cloud* são então notificados conforme seus registros como fornecedores dos tipos de recursos solicitados. Cada provedor decide se pode ou não atender tal demanda, respondendo diretamente ao *broker*, assim como reservando os recursos por um tempo determinado. O uso dessa técnica permite que o número de mensagens trocadas seja reduzido e garante que os recursos sejam alocados em determinado prazo, devido às reservas feitas.

5. Conclusão

Neste trabalho foi apresentada uma abordagem para a federação de provedores de *cloud* na qual foram descritos um modelo e os algoritmos de interação entre as entidades envolvidas. Adotamos um modelo de federação que faz uso de uma entidade global responsável pela busca e alocação de recursos em nome dos usuários na federação.

Novas abordagens foram propostas neste modelo de federação. Primeiramente, quanto à busca de recursos na federação, as demandas são postas em forma de leilão em

um Painel de Recursos. Os provedores somente respondem às demandas se podem atendê-las. Em caso contrário, não se manifestam nas negociações. Essa forma de interação torna mais simples a busca de recursos se compararmos com situações como a de repositórios globais ou serviços de informações, que dependem de atualizações frequentes feitas pelos provedores de *cloud*. Outro diferencial da abordagem proposta é quanto à inclusão de mecanismos de segurança no modelo de federação. Dos trabalhos apresentados na literatura, somente em [Celesti et al. 2010] são apresentados conceitos de autenticação e autorização para acesso a recursos na federação, porém sem a definição dos protocolos necessários e suas implicações. O modelo proposto emprega controles de segurança baseados no modelo de gerenciamento de identidades centralizado, que foi adotado por ser mais simples e adequado à proposta de federação descrita neste artigo.

Agradecimentos

Agradecemos a CAPES pelo apoio financeiro com bolsa de doutorado e bolsa CNPQ Processo [233648/2014-3].

Referências

Buyya, R., Ranjan, R. and Calheiros, R. (2010). Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. International Conference on Algorithms and Architectures for Parallel Processing, p. 13–31.

Calheiros, R. N., Ranjan, R., Beloglazov, A. and Rose, A. F. De (2011). CloudSim : a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. n. August 2010, p. 23–50.

Carlini, E., Coppola, M., Dazzi, P., Ricci, L. and Righetti, G. (2012). Cloud Federations in Contrail. Euro-Par 2011 Workshops, p. 159–168.

Celesti, A., Tusa, F., Villari, M. and Puliafito, A. (2010). How to Enhance Cloud Architectures to Enable Cross-Federation. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on.

Coppola, M., Dazzi, P., Lazouski, A., et al. (2012). The Contrail approach to cloud federations. The Internation Symposium on Grids and Clouds (ISGC) 2012,

Grozev, N. and Buyya, R. (2012). Inter-Cloud architectures and application brokering: taxonomy and survey. Software: Practice and Experience, p. 1–22.

Jøsang, A., Fabre, J., Hay, B., Dalziel, J. and Pope, S. (2005). Trust requirements in identity management. CRPIT '44: Proceedings of the 2005 Australasian workshop on Grid computing and e-research, p. 99–108.

Landwehr, C. E. (2001). Computer security. International Journal of Information Security, v. 1, n. July, p. 3–13.

Miller, J. (2006). Yadis Specification 1.0.

Rochwerger, B., Breitgand, D., Levy, E., et al. (jul 2009). The Reservoir model and architecture for open federated cloud computing. IBM Journal of Research and Development, v. 53, n. 4, p. 4:1–4:11.

Villegas, D., Bobroff, N., Rodero, I., et al. (sep 2012). Cloud federation in a layered service model. Journal of Computer and System Sciences, v. 78, n. 5, p. 1330–1344.