Controlo de Acessos em Sistemas com Consistência Fraca

Tiago Costa, Albert Linde, Nuno Preguiça e João Leitão

NOVA LINCS & DI-FCT-Universidade Nova de Lisboa

Resumo O modelo de consistência eventual tornou-se popular nos últimos anos em sistemas de gestão de dados em ambientes *cloud*. Neste modelo, é possível as réplicas estarem temporariamente inconsistentes, tendo sido propostas várias soluções para lidar com esta inconsistência e garantir a convergência final dos dados. A implementação de políticas de controlo de acessos para estes sistemas levanta desafios próprios, dado que a informação sobre permissões deve ser ela própria mantida de forma fracamente consistente. Neste artigo propõe-se uma solução para este problema prevenindo o acesso e modificação não autorizada dos dados. A solução proposta permite modificações concorrentes das políticas de segurança, garantindo a convergência das mesmas enquanto são usadas para executar o controlo de acessos aos dados associados. Neste artigo apresentamos uma avaliação inicial do modelo proposto, apresentando que a solução desenvolvida está de acordo com a verificação do funcionamento correto sobre possíveis situações problemáticas.

Keywords: Consistência eventual; Controlo de Acessos; Replicação

1 Introdução

Nos últimos anos os serviços Web tornaram-se centrais para a criação de software, suportando funcionalidades não apenas em aplicações Web mas generalizadamente em todo o software. Fatores importantes para o sucesso de aplicações que usam estes serviços são a disponibilidade do serviço e a latência das operações. Para suportar estes serviços com elevada disponibilidade e baixa latência é comum executar estes serviços em infraestruturas de cloud compostas por múltiplos centros de dados distribuídos geograficamente. Adicionalmente, é comum recorrer a sistemas de gestão de dados geo-replicados com um modelo de consistência fraca, nos quais as várias réplicas dos dados podem divergir momentaneamente garantindo o sistema apenas a convergência final dos dados.

As aplicações normalmente guardam informação sensível de utilizadores, os quais não pretendem que a mesma seja obtida sem autorização por outros utilizadores. O controlo de acessos à informação é assim um aspeto importante em qualquer aplicação com múltiplos utilizadores, sendo necessário definir políticas de controlo de acessos que especifiquem quem tem direito a aceder ou modificar cada parte da informação existente em cada momento. Estas políticas podem

mudar ao longo do tempo, sendo necessário que os mecanismos de controlo de acessos dos sistemas garantam que as mesmas tomam efeito a partir do momento em que são alteradas.

O controlo de acessos é um problema bastante estudado em sistemas de consistência forte, existindo múltiplas propostas de modelos e mecanismos para restringir o acesso à informação [15,7,6,5,9,14]. Estes modelos tipicamente assumem a existência duma ordenação total das operações que são executadas. Por exemplo, para controlar o acesso à informação numa rede social, ordenando as operações que modificam a lista de amigos e de acesso à informação dum utilizador, pode-se garantir facilmente que um utilizador não acede à informação de outro utilizador após ter sido removido da sua lista de amigos.

Nos sistemas com consistência fraca, não existe uma ordenação total das operações executadas no sistema. Assim, diferentes réplicas podem executar as operações por diferentes ordens. Este facto levanta novos desafios tanto para os modelos de controlo de acessos dos sistemas como para os mecanismos que implementam esses modelos.

Para exemplificar estes desafios, considere-se o contexto duma rede social em que um utilizador, Alice, tem inicialmente três amigos, Bob, John e Philip, e um álbum com uma fotografia, *beach.png*. Se a Alice pretender adicionar uma nova fotografia que não seja visível pelo Bob, pode removê-lo da sua lista de amigos e posteriormente adicionar a fotografia. Num sistema com consistência fraca, em que a lista de amigos e as fotografias dum álbum são armazenadas em diferentes objetos é possível que estas operações sejam propagadas de forma independente e executadas por uma ordem diferente em algumas réplicas, permitindo que o Bob aceda à nova fotografia. Como é conhecido, este problema concreto pode ser endereçado recorrendo a um modelo de consistência causal [11] que garante que a alteração da lista de amigos é sempre conhecida no momento em que se acede à nova fotografia.

Considere-se agora o caso em que qualquer amigo pode adicionar uma foto ao álbum da Alice. Neste caso, a adição de uma fotografia por um amigo (e.g. John) que concorrentemente é removido da lista de amigos (e.g. John) coloca a questão de qual deve ser o estado final do sistema. Esta problemática foi anteriormente endereçada em trabalhos anteriores no contexto de sistemas par-a-par [18] e *cloud* [17].

Neste artigo apresentamos um modelo de controlo de acessos para sistemas de gestão de dados que usem consistência fraca e mecanismos para implementar esse modelo. Como na solução proposta por Webber et. al. [17], a nossa solução permite a modificação concorrente dos dados e das políticas que controlam o acesso a este dados, definindo de forma determinista o comportamento do sistema. Ao contrário desse sistema, propomos um novo mecanismo de implementação do modelo definido que permite a sua utilização em sistemas de gestão de dados que não providenciem consistência causal, o que é comum na maior parte dos sistema NoSQL - e.g. Cassandra, Riak.

O artigo está organizado da seguinte forma: a secção 2 apresenta uma visão geral do funcionamento do sistema; na secção 3 discutimos com maior detalhe

os vários problemas que se levantam e damos algumas introduções a como se resolvem; a secção 4 discute detalhes de implementação; a secção 5 apresenta a nossa avaliação experimental; o trabalho relacionado é abordado na secção 6 e finalmente, a secção 7 conclui o artigo e discute brevemente direções futuras para o trabalho apresentado.

2 Modelos

Nesta secção apresentamos uma perspetiva geral do sistema proposto. Começamos por definir o modelo do sistema e de seguida apresentamos a forma de definir as políticas de segurança e o modelo de confiança do sistema.

2.1 Modelo do Sistema

Para discutirmos o funcionamento de um sistema de controlo de acessos com consistência fraca, vamos começar por definir o modelo do sistema. O modelo base consiste num sistema de gestão de dados que gere um conjunto de objetos, $Objs = \{o_1, o_2, \ldots\}$. O estado dos objetos pode ser modificado pela execução dum conjunto de operações, $Ops = \{op_1, op_2, \ldots\}$. Por simplicidade vamos assumir que uma operação de modificação altera o estado de um objeto. A função $target(op)$ devolve o objeto modificado pela operação *op*. As operações podem ser divididas em operações de leitura, Ops_{leit} e operações de escrita, Ops_{esc}.

Todas as operações são realizadas por um sujeito. O tuplo (op, s) é usado para representar a execução da operação *op* pelo sujeito s. O conjunto de sujeitos do sistema é designado por $Sujs = \{s_1, s_2, \ldots\}$.

Como é comum nos sistemas de *cloud*, assumimos que o sistema de gestão de dados mantém réplicas de todos os objetos em múltiplos centros de dados. Assim, para cada objeto existe uma réplica (ou mais) em cada centro de dados. Assumimos um modelo com consistência fraca com convergência final dos dados.

Um utilizador altera o estado dum objeto executando operações na réplica do objeto à qual está ligado. A esta execução na réplica inicial chamamos *upstream operation*. As operações são assincronamente propagadas de forma fiável para as outras réplicas, onde são executadas nas réplicas dos objetos. A esta execução nas restantes réplicas chamamos *downstream operations*. Apenas as operações que modificam o estado dos objetos são propagadas para as outras réplicas.

O estado dum objeto em cada réplica resulta da execução das *upstream operations* executadas localmente e das *downstream operations* recebidas de outras réplicas. Assim, na presença de operações concorrentes, diferentes réplicas executam as operações por diferentes ordens. Neste caso assumimos que o sistema garante a convergência final das réplicas, na qual todas as réplicas terão o mesmo estado após executarem o mesmo conjunto de operações. Em particular assumimos que o nosso sistema de gestão de dados usa CRDTs (*conflict-free replicated data types*) [16] para garantir a convergência final das réplicas.

Apesar de não existir uma ordem total entre as operações, podemos definir uma ordem parcial, aconteceu antes, que estabelece a potencial causalidade entre

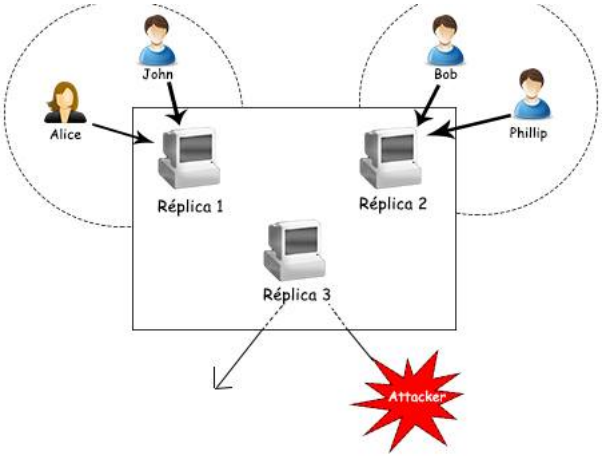


Figura 1: Rede de Confiança no Sistema

as operações. Dizemos que a operação op_a aconteceu antes de op_b, $op_a \rightarrow op_b$ sse os efeitos da operação op_a eram conhecidos na réplica a que o utilizador estava ligado quando executou a operação op_b. Diz-se que duas operações, op_a e op_b, são concorrentes, $op_a || op_b$ sse $op_a \not\rightarrow op_b \wedge op_b \not\rightarrow op_a$.

2.2 Políticas e Modelo de confiança

A política de segurança define as operações que cada utilizador pode executar (num dado momento). A atribuição de direitos a um determinado sujeito resulta num triplo $(r, s, o) \in Dirs \times Sujs \times Objs$ que indica que o sujeito s tem o direito r sobre o objeto o.

Representamos que o direito r permite executar a operação op por $r \vdash op$. No nosso sistema, por simplicidade, definimos apenas dois direitos: o direito a executar operações de leitura, r_l, e o direito a executar operações de escrita, r_e, com $\forall op \in Ops_{leit}, r_l \vdash op$ e $\forall op \in Ops_{esc}, r_e \vdash op$.

Um direito (r, s, o) permite ao sujeito s' executar uma operação op, $(r, s, o) \models (op, s')$ sse $r \vdash op \wedge target(op) = o \wedge s = s'$.

A informação relativa à política de segurança, i.e., os triplos com a atribuição de direitos, é mantida no sistema de gestão de dados e atualizada de forma semelhante aos dados. Assim, um utilizador atribui um novo direito executando uma operação na réplica à qual está ligado, sendo a operação propagada assincronamente para as outras réplicas.

Em relação ao grau de confiança nas réplicas do sistema, assumimos que todas as réplicas executam o modelo de segurança de acordo com o definido. O modelo de segurança que propomos baseia-se na verificação das permissões no momento da submissão das operações. Assim, quando um utilizador contacta uma réplica para executar uma operação, esse réplica é responsável por verificar que o utilizador tem permissões para executar essa operação, i.e., a execução da *upstream operation* é verificada. A execução duma *downstream operation* é

realizada sem a necessidade de qualquer tipo de verificação em relação às políticas atuais.

Esta aproximação é apropriada para um sistema de gestão de dados na *cloud* dado que se assume que todas as réplicas do sistema executam no contexto duma organização que as controla, pelo que não é expectável que as mesmas se comportem de forma maliciosa.

A Figura 1 mostra este nosso sistema, em que todas as réplicas podem comunicar diretamente sem necessitarem de verificar as permissões das operações. Quando um utilizador se autentica no sistema, o mesmo vai entrar temporariamente dentro do ambiente protegido, sendo que depois quando quiser executar alguma operação no sistema (*upstream operations*), esta operação vai ser verificada pela réplica à qual ele está ligado. No exemplo da figura, a Alice iria tentar fazer uma modificação mas só o conseguiria se na réplica 1 lhe fosse permitido executar a operação correspondente.

Desta forma qualquer utilizador não autenticado iria ficar sempre de fora do ambiente protegido. Adicionalmente, um utilizador autenticado que tentasse fazer escritas ou leituras para as quais não tinha permissão, veria estas operações serem rejeitadas.

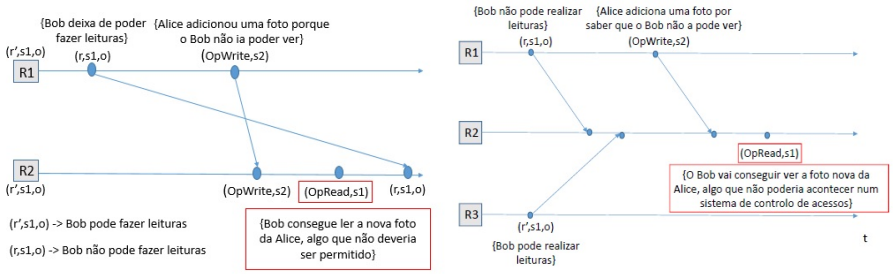
3 Semânticas do Controlo de Acesso e Desafios

Para um sistema de controlo de acessos estar correto, a política de controlo de acesso deve ser seguida em todas as réplicas. Num sistema com consistência forte, que emula o comportamento dum sistema com apenas uma réplica, uma operação op realizada pelo utilizador s é aceita se, aquando da sua execução existir um direito (r, s, o) que permite executar a operação, i.e., se $(r, s, o) \models (op, s)$.

Num sistema com consistência fraca é possível executar concorrentemente operações em diferentes réplicas. Apesar de assumirmos que o sistema garante a convergência final do estado das réplicas dos objetos na presença de operações concorrente, coloca-se a questão de decidir qual deve ser o comportamento do sistema na presença da execução concorrente de operações que modificam as permissões dum utilizador e de operações desse utilizador. De seguida vamos considerar as alternativas que se colocam, considerando separadamente as várias situações relevantes para o modelo de controlo de acessos.

Operações de Leitura: As operações de leitura são executadas apenas na réplica à qual o utilizador está conectado. Esta propriedade é fundamental para garantir uma baixa latência na execução da operação e uma elevada disponibilidade. Assim, para manter estas propriedades a única solução razoável parece ser a de controlar as permissões na réplica na qual o operação é executada com base nos direitos conhecidos localmente.

Caso concorrentemente, numa outra réplica, tenha sido executada uma operação que altera as permissões do utilizador, esta operação não terá efeitos na execução da operação de leitura. Para que as operações concorrentes tivesse efeito seria necessário efetuar uma coordenação com todas as réplicas antes de executar



(a) Violação de políticas de acesso caso 1 (b) Violação de políticas de acesso caso 2

Figura 2: Exemplo de execuções com manipulação de políticas.

qualquer operação, o que derrotaria as vantagens dos sistemas de consistência fraca.

Operações de Escrita: Um operação de escrita é executada na réplica à qual o utilizador está ligado como *upstream operation* e em todas as outras réplicas como *downstream operation*. Ao contrário das operações de leitura, neste caso, seria possível executar as operações de escrita e reverter os seus efeitos caso tivesse sido executada concorrentemente uma operação que impedisse a sua execução. No entanto, esta aproximação levaria a que fosse possível observar numa réplica o efeito duma operação que mais tarde seria revertida. Assim, na nossa aproximação, as permissões de execução duma operação de escrita são apenas verificadas no momento da sua execução inicial (i.e., quando são executadas como *upstream operation*).

Causalidade entre a execução de operações: Num sistema com controlo de acessos é importante garantir que as relações de causalidade entre as operações de modificação da política controlo de acessos e as operações sobre os dados são respeitadas. Considere-se o exemplo da Figura 2a, em que numa réplica a Alice executa uma operação para impedir o utilizador *Bob* de aceder às suas fotografias antes de adicionar uma nova fotografia. Se estas duas operações forem propagadas de forma independente e o sistema não garantir consistência causal no acesso aos dados, será possível nua outra réplica que o *Bob* acabe por ter acesso à fotografia, como se exemplifica na figura.

Assim, é fundamental que se garantam as relações de causalidade entre as operações que modificam a política de segurança relativa a um objeto e as operações de acesso a esse objeto. O nosso sistema garante esta propriedade, como se detalhará na próxima secção.

Modificações concorrentes às políticas de controlo de acessos: Além de ser possível executar concorrentemente operações que acedem aos dados e que alteram as permissões dos utilizadores, é igualmente possível executar concorrentemente operações que modificam as permissões dos utilizadores. Neste caso é necessário definir qual a política de controlo de acesso que deve ser mantida.

No nosso sistema usamos uma aproximação conservadora que privilegia as políticas de segurança mais restritivas. Assim, na presença de duas modificações

concorrentes à política de controlo de acesso, após ambas terem sido recebidas numa réplica, a que se manterá é a mais restritiva. Esta aproximação permite evitar os problemas de acessos indevidos como o que é apresentado no exemplo da Figura 2b. Neste exemplo, a Alice, após impedir o Bob de aceder às suas fotografias, adiciona uma nova fotografia. Se concorrentemente se permitisse ao Bob aceder às fotografias da Alice e em caso de alterações concorrentes prevalecesse a alteração mais permissiva, seria possível ao Bob aceder à fotografia.

Obviamente, a aproximação usada não garante que se obtém sempre o efeito pretendido. No entanto, com esta aproximação estamos a privilegiar a segurança do sistema. Estamos atualmente a estudar todas as implicações de permitir que a decisão de qual a política de segurança a manter seja definida pelo programador da aplicação (ou mesmo pelo utilizador).

Em resumo, e de forma mais formal, a execução de uma operação *op* pelo utilizador s no objeto o será permitida sse $\exists (r, s, o) \in H : r \vdash op \wedge \nexists (r', s, o) \in H : ((r, s, o) \rightarrow (r', s, o) \vee (r, s, o) || (r', s, o)) \wedge r' \nvdash op$, com H o conjunto de operações conhecida na réplica em que a operação é executada inicialmente.

4 Detalhes de Implementação

Como foi explicado na secção anterior, existiam dois desafios principais. O facto de poderem existir acessos não permitidos devido à propagação independente das modificações da política de controlo de acessos e de acesso aos dados, e como lidar com múltiplas modificações à política de controlo executadas de forma concorrente.

Implementámos a nossa solução sobre o sistema Antidote [1], um sistema de gestão de dados distribuído que funciona sob o modelo consistência fraca com convergência final dos dados. Toda a gestão de dados desse sistema recorre ao uso de CRDTs.

De forma a serem implementadas políticas de controlo de acesso foram desenvolvidos novos CRDTs que incluem informação explícita relativa ao controlo de acessos. Estes CRDTs mantêm as propriedades genéricas do CRDTs, i.e., garantem a convergência final do estado das réplicas na presença de modificações concorrentes. Os nossos CRDTs dispões de dois métodos relevantes à sua operação: a função de *generateDownstream*, que gera uma operação remota a ser executada nas restantes réplicas; e a função *update*, que é chamada sempre que uma réplica recebe uma operação de *downstream* (i.e, uma operação remota).

Para implementar o nosso modelo de segurança, no nosso sistema, a operação de *downstream* inclui o valor das políticas de controlo de acesso na réplica em que a operação foi submetida. A função *update* foi estendida para receber, como argumento, o valor das políticas local assim como o valor remoto. No caso em que a política local e remota são distintas, além de executar a operação é necessário atualizar a política de controlo de acessos local.

Vamos agora explicar como esta aproximação permite responder aos desafios indicados anteriormente. Em primeiro lugar, temos como requisito evitar o acesso indevido na presença de operações que atualizam as políticas de controlo

de acessos de forma concorrente. Quando existem modificações concorrentes a representação de cada uma delas vai dar-nos o conhecimento sobre o que cada réplica permite, sendo que o objetivo é existir uma defesa contra a possibilidade de serem feitas posteriormente edições ou leituras sobre os dados que deveriam estar protegidos. No exemplo da Figura 2b, seria necessário registar na réplica intermédia que estão a ocorrer operações concorrentes e que as mesmas tinham de ser consideradas no conjunto sem descartar nenhuma delas. Conhecendo todas as operações que modificam as políticas de segurança concorrentemente, é possível optar pela opção mais segura selecionando a permissão mínima (conhecida) para cada utilizador.

O desafio seguinte está relacionada com a ordenação das operações (Figura 2a. Para suportar esta funcionalidade, podia ter sido usada como solução requerer causalidade na propagação de operações entre réplicas para garantir que uma operação de acesso aos dados apenas é executada após qualquer operação de atualização de políticas que tenha ocorrido (logicamente) antes. No entanto esta solução têm limitações notórias no caso de operações que manipulam múltiplos objetos.

A solução adotada passa pela adição das políticas de controlo de acesso a todas as operações *downstream*. Quando é realizada uma operação de escrita as políticas locais na réplica que recebe essa operação são propagadas explicitamente no pedido downstream correspondente. Recorrendo novamente ao exemplo da Figura 2a, se a Alice remove a capacidade de leitura do Bob e depois faz uma operação de escrita (e.g., adicionar a foto), essa operação de escrita será propagada *downstream* conjuntamente com a política local da réplica R1, ou seja, irá conter explicitamente a informação de que Bob não pode fazer leituras. Quando essa informação chega à réplica R2 a escrita da Alice vai ser efetuada ao mesmo tempo que as políticas de acesso em R2 são atualizadas com essa informação. Desta forma quando o Bob tenta fazer a operação de leitura vai ser calculado o mínimo das suas permissões, sendo que esse mínimo irá refletir o facto de Bob ter perdido direitos de leitura, levando o seu pedido a ser recusado.

5 Avaliação

Nesta secção validamos a nossa abordagem através da execução de um conjunto de testes de correção. Nestes testes utilizamos temos uma única réplica do sistema de armazenamento a executar, e simulamos várias outras réplicas que geram operações downstream para esta réplica (as restantes réplicas são emuladas recorrendo a ferramentas de teste do sistema Antidote).

Estes testes foram realizados usando várias réplicas a produzirem operações downstream para uma réplica apenas, para desta forma conseguirmos ver o seu estado local e aferir os resultados.

A avaliação foi realizada usando os CRDTs seguros descritos neste artigo e um tipo de dados normais para uma base de comparação. Em ambos dizemos qual o resultado esperado num bom funcionamento de um sistema de controlo de

Tabela 1: Teste Ordenação de Operações

Réplica	Ator	Operação	Valor do Objeto na Réplica
1-R1 (Op1)	Alice	Mudança Policica: {"Bob", []}	{0, 0, [{“Alice”„[read, write, writeplus, own]}, {“Bob”, [read, write]}]}
2-R1	Update	Update de 1	{0, 0, [{“Alice”„[read, write, writeplus, own]}, {“Bob”, []}]}
3-R1 (Op2)	Alice	Incremento: 3	{0, 0, [{“Alice”„[read, write, writeplus, own]}, {“Bob”, []}]}
4-R1	Update	Update de 3	{3, 0, [{“Alice”„[read, write, writeplus, own]}, {“Bob”, []}]}
5-R2	Update	Update de 3	{3, 0, [{“Alice”„[read, write, writeplus, own]}, {“Bob”, [???]}]}
6-R2 (Op3)	Bob	Leitura sobre objeto	{3, 0, [{“Alice”„[read, write, writeplus, own]}, {“Bob”, [???]}]}
7-R2	Update	Update de 1	{3, 0, [{“Alice”„[read, write, writeplus, own]}, {“Bob”, [???]}]}

Tabela 2: Teste Ordenação de Operações Asserções

Operação	Asserção	Resultado Secure CRDT	Resultado Tipo de Dados Normal
5	AssertEqual value(Bob) == []	Ok	AssertEqualFailed - expected []; value(Bob) = [read,write]
6	AssertEqual value(Bob) == []	Ok	AssertEqualFailed - expected []; value(Bob) = [read,write]

acesos e comparamos esse resultado com asserções para aferir se o funcionamento correto está a ser assegurado.

O primeiro teste valida a ordenação de operações. Para este teste foram usadas apenas 2 réplicas, tal como representado na figura 3 (1). O estado inicial do nosso objeto é {0, 0, [{“Alice”, [read, write, writeplus, own]}, {“Bob”, [read, write]}]}.

Neste caso pretendemos verificar se Bob consegue observar a imagem adicionada por Alice depois de esta ter removido a capacidade de leitura de Bob. Tal como indicado na Figura, Alice executa as operações sobre a réplica R1 e Bob sobre a réplica R2. Como discutido anteriormente, o resultado esperado é que a operação de leitura de Bob não reflita a imagem adicionada por Alice. As asserções utilizadas para verificar este caso e os resultados obtidos recorrendo à nossa solução e aos CRDTs nativos do sistema Antidote encontram-se descritos na Tabela 2.

No segundo teste, recorremos ao exemplo ilustrado na Figura 3 (2) em que existe uma modificação concorrente das políticas de leitura de Bob (réplicas R1 e R3). Neste exemplo Alice adiciona uma imagem após remover a capacidade de leitura de Bob em R1, e Bob tenta observar o estado em R2. O estado inicial do objeto alvo é 0, 0, [“Alice”, [read, write, writeplus, own], “Bob”, [read,write], “John”, [read, write, writeplus, own].

Neste caso a questão é o valor presente na réplica R2 após a execução da sexta operação. Como discutido anteriormente, o resultado esperado implica que Bob não consiga observar a imagem adicionada por Alice, visto que antes desta operação Alice remove a capacidade de leitura de Bob. A Tabela 4 indica as asserções utilizadas para testar este caso assim como os resultados obtidos recorrendo à nossa solução e utilizando os CRDTs nativos do sistema Antidote.

Tabela 3: Teste Políticas Concorrentes

Réplica	Ator	Operação	Valor do Objeto na Réplica
1-R1 (Op1)	Alice	Mudança politica: {"Bob", []}	{0, 0, [{"Alice",„[read, write, writeplus, own]}, {"Bob", [read, write]}]}
2-R1	Update	Update de 1	{0, 0, [{"Alice",„[read, write, writeplus, own]}, {"Bob", []}]}
3-R3 (Op2)	John	Mudança politica: {"Bob", [read]}	{0, 0, [{"Alice",„[read, write, writeplus, own]}, {"Bob", [read, write]}]}
4-R3	Update	Update de 3	{0, 0, [{"Alice",„[read, write, writeplus, own]}, {"Bob", [read]}]}
5-R1	Alice	Incremento: 3	{0, 0, [{"Alice", [read, write, writeplus, own]}, {"Bob", []}]}.
6-R1	Update	Update de 5	{3, 0, [{"Alice", [read, write, writeplus, own]}, {"Bob", []}]}.
7-R2	Update	Update de 1	{0, 0, [{"Alice",„[read, write, writeplus, own]}, {"Bob", []}]}
8-R2	Update	Update de 3	{0, 0, [{"Alice",„[read, write, writeplus, own]}, {"Bob", [???]}]}
9-R2	Update	Update de 5	{3, 0, [{"Alice", [read, write, writeplus, own]}, {"Bob", [???]}]}.
10-R2 (Op3)	Bob	Leitura sobre objeto	{3, 0, [{"Alice",„[read, write, writeplus, own]}, {"Bob", [???]}]}

Tabela 4: Teste Políticas Concorrentes Asserções

Operação	Asserção	Resultado Secure CRDT	Resultado Tipo de Dados Normal
8	AssertEqual value(Bob) == []	Ok	AssertEqualFailed - expected []; value(Bob) = [read]
10	AssertEqual value(Bob) == []	Ok	AssertEqualFailed - expected []; value(Bob) = [read]

Como mostrado nas Tabelas 2 e 4, ao utilizar a nossa solução o sistema exibe sempre o comportamento esperado, em que Bob não consegue observar a imagem introduzida por Alice. Contrariamente, ao usar os CRDTs nativos do sistema Antidote, que não implementam um mecanismo de controle de acesso, Bob é capaz de observar esta imagem, demonstrado claramente os perigos de privacidade inerentes a sistemas de armazenamento de dados que operam sobre o modelo de consistência eventual.

6 Trabalho Relacionado

Na área de controlo de acessos em sistema com consistência eventual, existem dois tipos de trabalhos relevantes: sistemas de armazenamento de dados e ferramentas de edição colaborativa.

Sistemas de armazenamento de dados com consitência eventual Apesar de muitos protocolos e pesquisa terem sido feitos sobre Data stores com consistência eventual, pouco ênfase tem sido dado ao controlo de acessos neste tipo de sistemas. A versão original do Amazon Dynamo [4] não oferecia nenhum tipo de autenticação ou autorização. Outros sistemas de armazenamento com consistência eventual já oferecem algumas técnicas de acesso de controlo mas com pouca granularidade, insuficiente para providenciar controlo ao nível aplicacional. Couchbase [3], MongoDB [12] e Riak KV [13] todos suportam utilizadores, funções e permissões, mas com alta granularidade, ao nível dos *buckets* e coleções de dados. Contrariamente, a nossa solução consegue operar ao nível de cada objeto individual.

Wobber et al. [18] apresenta um controlo de acessos para consistência fraca em sistemas onde não existe confiança mútua. O escopo é diferente pois consiste numa replicação apenas parcial com diferentes acessos para cada réplica. Este modelo também apresenta problemas similares em relação à causalidade entre as políticas e operações subsequentes que sejam permitidas. No seu modelo existe uma espera pela política desejada, sendo que mesmo assim a causalidade entre mudanças de

políticas que restringem a visibilidade de efeitos de operações e as subsequentes execução de operações não é capturada. Devido a isso, e contrariamente à nossa solução, este mecanismo ainda permite a observação indevida de informação por parte dos utilizadores.

Alguns sistemas de armazenamento de dados geo-replicados que oferencem garantias de consistência causal como o COPS [11] ou o ChainReaction [2] têm parte da sua motivação associada a um exemplo similar ao que usamos neste artigo. Estes trabalho no entanto assumem que a lista de controle de access de cada objeto é ela própria um objeto independente no contexto do sistema de armazenamento de dados, e consequentemente a propagação causal de operações entre réplicas localizadas em centros de dados distintos evitam anomalias em que os utilizadores podem observar estado ao qual não deveriam ter acesso. No entanto o custo necessário para garantir consistência causal não é desprezável, podendo levar o sistema a ver o seu desempenho significativamente degradado. Em contraste, a nossa proposta não requer garantias extra do ponto de vista do modelo de consistência, evitando assim esses custos adicionais.

Ferramentas de Edição Colaborativa Imine et al. [8] apresenta um modelo de acesso de controlo para edição colaborativa distribuída. As modificações dos documentos no editor são vistas como operações sobre dados sendo que o documento precisa de consistência eventual em todos os editores distribuídos. Para prevenir divergências, todas as operações num objeto são ordenadas pelo autor do editor. Isto implica uma relação de um para um entre parte do texto e o editor responsável, o que leva a um ponto possível de falha. Também este modelo apenas considera operações de modificação, sendo todos os utilizadores permitidos a ler o documento completo. No nosso caso temos uma restrição das leituras o que nos permite falar sobre fugas de informação.

7 Conclusão e Trabalho Futuro

Neste artigo vamos de encontro a estabelecermos controlo de acessos a dados geo-replicas num sistema de gestão de dados centralizado que usa consistência eventual. O próximo passo será estender estas políticas de controlo de acesso para conseguir uma correta (e confiável) execução na situação onde são os próprios clientes a replicar dados entre si. O sistema Legion [10] procura enriquecer a forma como são vistas as aplicações web, oferecendo uma interação direta entre clientes para aumentar a disponibilidade e diminuir latência na propagação de operações, criando uma rede de *peer-to-peer* entre clientes. Como as restantes réplicas do sistema (outros clientes) não podem ser admitidos como confiáveis, torna-se necessário verificar as atualizações recebidas de outras réplicas (i.e., mensagens passadas entre clientes não pertencem à base de confiança visto não podermos confiar em quem envia e/ou efetuou uma operação, mas podemos alavancar no sistema centralizado apresentado neste artigo).

Referências

1. Akkoorath, D.D., Tomsic, A.Z., Bravo, M., Li, Z., Crain, T., Bieniusa, A., Preguiça, N., Shapiro, M.: Cure: Strong semantics meets high availability and low latency. In: Proc. 36th IEEE International Conference on Distributed Computing Systems (ICDCS 2016) (Jun 2016)
2. Almeida, S., Leitão, J., Rodrigues, L.: Chainreaction: A causal+ consistent datastore based on chain replication. In: Proceedings of the 8th ACM European Conference on Computer Systems. pp. 85–98. EuroSys '13 (2013)
3. Couchbase: Couchbase, http://www.couchbase.com/
4. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In: ACM SIGOPS Operating Systems Review. vol. 41, pp. 205–220. ACM (2007)
5. Ferraiolo, D., Kuhn, D.R., Chandramouli, R.: Role-based access control. Artech House (2003)
6. Hu, V.C., Ferraiolo, D., Kuhn, R., Friedman, A.R., Lang, A.J., Cogdell, M.M., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K., et al.: Guide to attribute based access control (abac) definition and considerations (draft). NIST Special Publication 800, 162 (2013)
7. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-based access control. Computer (2), 85–88 (2015)
8. Imine, A., Cherif, A., Rusinowitch, M.: A flexible access control model for distributed collaborative editors. In: Secure Data Management, pp. 89–106. Springer (2009)
9. Jin, X., Sandhu, R., Krishnan, R.: Rabac: role-centric attribute-based access control. In: Computer Network Security, pp. 84–96. Springer (2012)
10. Linde, A.: Enriching Web Applications with Browser-to-Browser Communication. Master's thesis, Faculdade Ciência e Tecnologia, Universidade Nova de Lisboa (7 2015)
11. Lloyd, W., Freedman, M.J., Kaminsky, M., Andersen, D.G.: Don't settle for eventual: Scalable causal consistency for wide-area storage with cops. In: Proc. of the 23rd ACM Symposium on Operating Systems Principles. pp. 401–416. SOSP '11 (2011)
12. MongoDB: Mongodb, https://www.mongodb.com/
13. Riak: Riak kv., http://basho.com/products/riak-kv/
14. Samarati, P., Di Vimercati, S.D.C.: Access control: Policies, models, and mechanisms. Lecture notes in computer science pp. 137–196 (2001)
15. Sandhu, R.S.: Role-based access control. Advances in computers 46, 237–286 (1998)
16. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Stabilization, Safety, and Security of Distributed Systems, pp. 386–400. Springer (2011)
17. Weber, M., Bieniusa, A., Poetzsch-Heffter, A.: Access control for weakly consistent cloud-storage systems (2016), submitted for publication
18. Wobber, T., Rodeheffer, T.L., Terry, D.B.: Policy-based access control for weakly consistent replication. In: Proceedings of the 5th European conference on Computer systems. pp. 293–306. ACM (2010)