C2FS: um Sistema de Ficheiros Seguro e Fiável para *Cloud-of-Clouds*

Ricardo Mendes, Tiago Oliveira, Marcelo Pasin, Alysson Bessani

Universidade de Lisboa, Faculdade de Ciências
{rmendes,toliveira}@lasige.di.fc.ul.pt,{pasin,bessani}@di.fc.ul.pt

Resumo Vários sistemas surgiram recentemente com o intuito de facilitar a utilização de serviços de armazenamento nas clouds. Muitos utilizadores das clouds têm a necessidade de manter os seus dados disponíveis e privados, requisitos nem sempre atendidos pelos sistemas de armazenamento em cloud. Recentemente foi demonstrado que é possível atender estes requisitos através do uso de vários provedores de cloud, ao invés de um só, ao que foi dado o nome de *cloud-of clouds*. Com vista a responder a estas necessidades apresentamos o C2FS, um sistema de ficheiros multi-utilizador para *cloud-of-clouds*. Este sistema é tolerante a falhas por parte dos provedores de clouds e mantém a privacidade dos dados e metadados armazenados desde que menos de um terço dos provedores usados sejam faltosos. O C2FS tem uma interface estilo POSIX e satisfaz um modelo de consistência flexível que permite aos seus utilizadores controlarem os custos (em termos monetários e de desempenho) relacionados com o acesso a clouds e as garantias de consistência e durabilidade oferecidas pelo sistema.
Palavras-chave: sistema de ficheiros, computação em clouds, armazenamento nas clouds, tolerância a faltas bizantinas.

1 Introdução

A crescente popularidade dos serviços de armazenamento em clouds tem vindo a permitir que utilizadores e empresas possam exteriorizar o armazenamento dos seus dados tanto para fins de *backup* quanto para facilitar acesso aos mesmos. Exemplos destes serviços são o DropBox, GoogleDrive, iCloud e Amazon S3. Uma das vantagens fundamentais destes serviços, para além da facilidade de acesso, é o seu baixo custo e o modelo de pagamento conforme o uso, que não requer investimento inicial. No entanto, existem também desvantagens no uso deste tipo de serviços. Em primeiro lugar o acesso aos dados pode ser impedido por períodos de indisponibilidade do provedor, ou pela ligação até ele. Além disso, há sempre a preocupação com o acesso irrestrito do provedor aos dados por ele armazenados, o que dificulta o armazenamento de informações sensíveis. Um perigo ainda mais grave reside no facto do provedor poder corromper os dados devido a faltas (maliciosas ou não) na sua infraestrutura. Finalmente, existe o perigo do serviço aumentar o seu preço ou até impedir o utilizador de remover os seus dados, situação esta conhecida como *vendor lock-in*.

Recentemente foi demonstrado que o uso de múltiplos provedores independentes, aliado a técnicas de tolerância a faltas e criptografia, permite reduzir

os riscos mencionados [5]. Neste âmbito foi apresentado o sistema DepSky, que permite armazenar objectos (blocos opacos de dados associados a uma chave de acesso) numa *cloud-of-clouds* composta por provedores diferentes de tal forma que os dados sejam acessíveis e privados mesmo que uma fracção dos provedores estejam sujeitos a faltas bizantinas (exibindo um comportamento arbitrário, fora da sua especificação). Porém, o DepSky apresenta uma interface de baixo nível, assemelhando-se a um disco virtual onde blocos de dados são armazenados. Este tipo de serviço é de difícil uso tanto para programadores (que constroem aplicações com armazenamento de dados na cloud) quanto para utilizadores, que apenas querem guardar seus ficheiros na cloud. Por outro lado, utilizadores e programadores tendem a sentir-se muito mais confortáveis com abstracções como a de sistema de ficheiros.

Neste artigo apresentamos o C2FS, um sistema de ficheiros para armazenamento de dados na *cloud-of-clouds*. O C2FS é compatível com a interface POSIX, pode ser montado em sistemas operativos Linux e, tal como o DepSky, tolera faltas por parte dos provedores. Para além disso, o C2FS oferece garantias de controlo de acesso a ficheiros partilhados e um nível de consistência forte (*consistência ao fechar* [8]).

Para oferecer um serviço com essas garantias, o C2FS usa dois outros sistemas como blocos de construção. O já mencionado DepSky [5] é usado como um "disco virtual" onde os ficheiros são armazenados de forma persistente na *cloud-of-clouds*. Além deste, o serviço de coordenação DepSpace [4], que é usado para armazenar os metadados dos ficheiros assim como para coordenar escritas concorrentes nos mesmos. O C2FS aproveita as propriedades de tolerância a faltas bizantinas e confiencialidade dos dados destes dois sistemas. A integração destes sistemas não é trivial, pelo que um dos grandes desafios deste trabalho é combinar esses sistemas provendo, ao mesmo tempo, uma semântica que faça sentido do ponto de vista de um sistema de ficheiros.

A principal contribuição deste artigo é apresentar o primeiro sistema de ficheiros com armazenamento em clouds que garante ao mesmo tempo (1) tolerância a problemas nos provedores de serviço de armazenamento, (2) partilha e controlo de acessos concorrentes a ficheiros, (3) confidencialidade dos dados armazenados nos provedores (nenhum provedor tem acesso aos dados armazenados), e (4) desempenho aceitável ao mesmo tempo que é garantida consistência forte (ao fechar), através do uso cuidadoso de cache tanto em memória quanto em disco local.

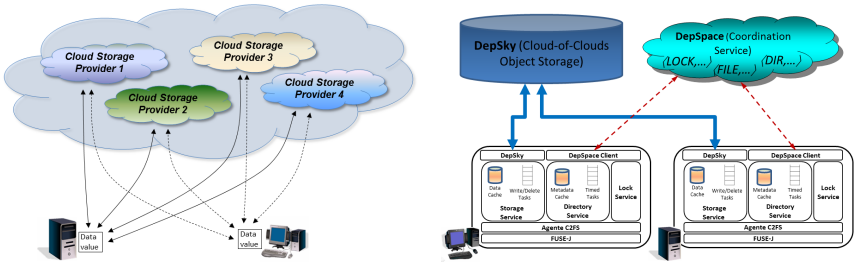
2 Arquitectura do Sistema

A figura 1 apresenta a arquitectura do C2FS e a estrutura dos seus serviços. Na figura 1(a) é apresentada uma visão geral do sistema: um conjunto de utilizadores que acedem a ficheiros armazenados em diversos provedores de serviços em clouds (computação e armazenamento). Já na figura 1(b) temos a estrutura dos seus componentes básicos. Na base da arquitectura encontra-se o FUSE [1], módulo responsável por interceptar as chamadas ao sistema operativo referentes aos ficheiros armazenados no C2FS. Estas chamadas de sistema são entregues

ao agente C2FS que tem como função interagir com os serviços de *locking*, de directorias e de armazenamento de modo a satisfazer cada chamada relacionada com o sistema de ficheiros feita pelo kernel.

O serviço de *locking* permite manter a consistência dos ficheiros, garantindo que, a cada instante, existe no máximo um utilizador a escrever em cada ficheiro, enquanto que os serviços de armazenamento e de directorias gerem os dados e metadados, respectivamente, assegurando a disponibilidade e confidencialidade destes. Os serviços de directorias e *locking* fornecem estas garantias devido ao uso do DepSpace[1] [4]. Estas garantias são também fornecidas pelo serviço de armanezamento recorrendo ao DepSky [5].

Como podemos ver na figura 1(b), tanto o serviço de directorias como o serviço de armazenamento mantêm uma cache e fazem gestão de uma colecção de tarefas que correm em *background*. Esta utilização permite ao C2FS atingir um maior desempenho (operando sobre os dados e metadados localmente) ao mesmo tempo que diminui o número de operações feitas tanto no DepSpace como na *cloud-of-clouds*. Este sistema de ficheiros só fornece garantias de consistência entre as caches e a *cloud-of-clouds* quando um ficheiro é fechado. As tarefas em *background* permitem a estes serviços efectuar actualizações de dados e metadados, não prejudicando a performance global do sistema.



(a) *Cloud-of-clouds.* (b) Componentes do C2FS.

Fig. 1. Arquitectura do C2FS.

Em seguida apresentamos uma breve discussão sobre os três componentes básicos usados na construção do C2FS, ressaltando as modificações e extensões necessárias para a sua utilização no nosso sistema.

FUSE. O FUSE (File System in Userspace) [1] é um módulo para o Linux que permite a construção de sistemas de ficheiros fora do kernel. O módulo (instalado no kernel) intercepta chamadas de acesso a recursos controlados pelo sistema FUSE (e.g., *open*, *write*) e envia-as para o processo que concretiza o sistema de ficheiros, que as executa e retorna uma resposta ao módulo FUSE. Tal como outros sistemas de ficheiros modernos [3,14], também o C2FS utiliza este serviço para implementar um sistemas de ficheiros em nível de utilizador. Devido a isto, é oferecida aos clientes uma interface estilo POSIX que torna

[1] Idealmente, as réplica do DepSpace deve estar a correr em VMs localizadas em diferentes provedores de computação em cloud, fazendo também uso do paradigma *clouds-of-clouds*.

transparente o armazenamento de dados na *cloud-of-clouds*. A escolha do FUSE justifica-se pela simplicidade que este oferece na prototipagem de um sistema de ficheiros. É usada uma concretização em Java do FUSE chamada FUSE-J, que facilita a interacção com outros componentes já existentes, como o DepSky e o DepSpace.

DepSpace. O DepSpace é um serviço de coordenação (nos moldes do Chubby [6] e ZooKeeper [9]) que implementa uma abstracção de espaço de tuplos e tolera faltas bizantinas [4]. Este serviço possibilita aos seus clientes armazenar estruturas de dados genéricas (chamadas tuplos) e realizar operações com poder de sincronização como o *cas* (*condicional atomic swap*), que possibilita a inserção de um tuplo no espaço apenas no caso de não existir nenhum outro tuplo no espaço que satisfaça um template dado. Fornece ainda a possibilidade de definir políticas de acesso aos tuplos contidos no espaço, nomeadamente definir quais os clientes do serviço que estão habilitados a ler ou alterar o conteúdo dos tuplos. Além disso, o serviço possibilita a criação de tuplos temporários: tuplos que permanecem no espaço por um tempo definido e, a menos que este seja revalidado (através da chamada da operação *renew*), são automaticamente removidos quando expirados.

Para a implementação do C2FS, e em especial de um serviço de directorias eficiente, foi necessário adicionar duas novas funcionalidades ao DepSpace. Uma das limitações do modelo de espaço de tuplos é a impossibilidade de modificar um tuplo no espaço. Esta limitação aparece para um serviço de directorias quando, por exemplo, é necessário renomear um nó da árvore de directorias, que é representado por um tuplo no DepSpace. A forma usual de se alterar tuplos é remover a versão antiga e adicionar uma nova. No entanto, esta solução levanta problemas tanto de desempenho (pelo menos duas operações no espaço) como de consistência (as duas operações não são atómicas). A solução para este problema foi modificar o DepSpace para suportar uma nova operação, chamada *replace*($\bar{t}$,t), que substitui o tuplo mais antigo no espaço que satisfaça $\bar{t}$ por t. Um segundo problema mais grave ocorre quando renomeamos uma directoria. Quando isso acontece, o campo "nó pai" de todos os nós dentro dessa directoria tem de ser actualizado. Para resolver este problema de maneira eficiente, i.e., evitar a execução de um *replace* por cada nó filho do nó renomeado, foi adicionada ao DepSpace uma nova camada para suporte a *Triggers* que, aquando da execução de uma operação *replace*, activa o gatilho para execução de um conjunto de outras operações nos servidores (e.g., substituição dos nós filhos)[2].

DepSky. O DepSky [5] é um sistema para armazenamento de dados fiável e seguro para *cloud-of-clouds*. Uma das grandes vantagens deste sistema prende-se com o facto de ser tolerante a faltas bizantinas, eliminando portanto a necessidade de se confiar nas clouds individualmente. Para além disso, o DepSky emprega primitivas de partilha de segredo e códigos de apagamento para garantir que nenhuma

[2] É utilizado o nome do nó pai ao invés do seu identificador pois as pesquisas feitas pelo sistema operativo são pelo ao nome dos nós. A utilização do identificador resultaria num acréscimo de chamadas ao DepSpace (resolução do nome para identificador).

cloud tem acesso ao conteúdo dos dados armazenados, garantindo também que o custo de se armazenar dados num conjunto de n clouds é proporcional a $\frac{n}{2}$. Este sistema dá-nos assim garantias de disponibilidade, integridade e confidencialidade dos dados, com um acréscimo de 100% no custo de armazenamento.

Assim como no caso do DepSpace, o DepSky teve de ser modificado para que atendesse os requisitos do C2FS. Por motivos relacionados com a partilha de ficheiros e com o controlo de acesso aos mesmos, o serviço de armazenamento (um componente do C2FS) cifra os dados antes de os guardar nas clouds (usando o DepSky), sendo a chave de acesso guardada no tuplo correspondente aos metadados do ficheiro (no DepSpace). Neste cenário não há necessidade do DepSky cifrar os dados de novo (e nem de usar partilha de segredo para garantir confidencialidade das chaves). Assim, foi criado uma variante da implementação corrente que mantém a disponibilidade e integridade dos dados, reduz os custos monetários de armazenamento devido à utilização de códigos de apagamento, mas não faz uso de cifra nem de partilha de segredos.

3 Serviço de Directorias

O C2FS usa um serviço de directorias que tem por base o DepSpace. Esta abordagem faz com que os metadados possam ser guardados utilizando a abstração de espaço de tuplos ao mesmo tempo que permite que o serviço herde todas as garantias fornecidas pelo DepSpace. Os metadados de todos os ficheiros, directorias e ligações simbólicas são representados como tuplos no espaço. Esses tuplos têm o seguinte formato:

$$\langle node\_type, parent\_folder, node\_name, node\_metadata, node\_id \rangle$$

Neste formato, o campo *node_ type* pode tomar o valor de FILE, DIR ou SYMLINK consoante os metadados sejam de um ficheiro, uma directoria ou uma ligação simbólica, respectivamente. O campo *node_ metadata* contém toda a informação dos metadados de um determinado nó, tais como as permissões dos vários utilizadores do sistema sobre ele, o identificador do dono do ficheiro, o tamanho do ficheiro, o resumo criptográfico dos dados a que este se refere e a chave simétrica utilizada para cifrar os dados. A importância de manter o resumo criptográfico e a chave neste serviço será explicada em detalhe mais à frente. O campo *node_ id* é o identificador único de um contentor dos dados de um nó dentro do sistema de ficheiros. Este identificador é criado pelo cliente do serviço de directorias que cria este contentor.

Este serviço de directorias exporta uma API que permite efectuar todas as operações necessárias para manipular os metadados do sistema. A tabela 1 apresenta uma lista dessas operações e a sua interacção com o DepSpace.

Como podemos perceber, para armazenar os metadados dos nós do sistema de ficheiros, o serviço de directorias insere tuplos no espaço. A operação usada para inserir tuplos nos espaço é a operação *cas* pois permite, de uma forma atómica, inserir um tuplo no espaço apenas no caso de não existir nenhum outro que corresponda ao mesmo nó do sistema de ficheiros.

O serviço de directorias fornece ainda um mecanismo que permite aos clientes controlar o acesso aos metadados. Este evita que clientes não autorizados (e.g.,

Op. do Serviço de Directorias	Op. do DepSpace
Get(parent, name)	rdp(⟨*, parent, name, *, *, *⟩);
Put(mdata)	cas(⟨*, parent(mdata), name(mdata), *, *, *⟩, ⟨nodeType(mdata), parent(mdata), name(mdata), mdata, nodeId(mdata)⟩);
Remove(parent, name)	inp(⟨*, parent, name, *, *, *⟩);
Update(parent, name, mdata)	replace(⟨*, parent(mdata), name(mdata), *, *, *⟩, ⟨nodeType(mdata), parent(mdata), name(mdata), mdata, nodeId(mdata)⟩);
GetChildren(parent)	rdAll(⟨*, parent, *, *, *, *⟩);
GetLinks(inode)	rdAll(⟨*, *, *, *, *, inode⟩);

Tabela 1. Interacção do serviço de directorias com o DepSpace.

que não tenham acesso a um nó) tenham acesso ao tuplo que representa o nó no DepSpace. Este mecanismo está assente sobre as políticas de controlo de acesso aos tuplos por parte do DepSpace.

Caching no serviço de directorias. O serviço de directorias pode realizar caching de metadados em memória local de forma a minimizar tanto a latência de acesso aos metadados[3], quanto o número de chamadas feitas ao DepSpace (permitindo a este atender mais clientes).

A cache de metadados do C2FS é concretizada utilizando tarefas temporizadas, responsáveis pela actualização dos metadados no DepSpace após a expiração de um temporizador. Estas tarefas são criadas com o objectivo de efectuar actualizações nos metadados de um determinado nó no DepSpace de forma assíncrona ao sistema e com um atraso esperado de no máximo Δ ms, sendo Δ um parâmetro do sistema. Este mecanismo permite também que todas as actualizações feitas aos metadados de um nó em cache durante um intervalo de Δ ms sejam executadas de uma única vez no DepSpace. O valor de Δ define também por quanto tempo os metadados de um determinado nó estão em cache no serviço de directorias. Assim, a definição deste parâmetro deve ter em conta que, quanto maior seu valor, menor o número de chamadas feitas ao DepSpace, e maior o atraso de propagação de alterações de metadados, o que pode ter efeitos negativos na consistência do sistema.

Para lidar com actualizações de metadados em consequência de actualizações de dados, o serviço permite modificar os metadados apenas na cache local, mantendo estas alterações por tempo indeterminado até que o cliente do serviço decida efectuar essas alterações de forma definitiva no DepSpace. Isto é usado, e.g., quando o cliente tem de fazer escritas demoradas de dados (geralmente feitas em *background*, ver secção 5). Neste caso o sistema primeiro efectua a escrita dos dados numa cache em disco local, depois altera os metadados na cache local e retorna ao sistema operativo, fazendo a alteração no DepSpace apenas quando a escrita dos dados terminar.

4 Serviço de Locking

Para manter a consistência dos ficheiros do C2FS, foi desenvolvido um serviço de *locking* que controla o acesso simultâneo a ficheiros. Tal como o serviço de directorias, também este foi desenvolvido tendo o DepSpace como base.

[3] Acedidos através das chamada *stat*, identificada em muitos estudos como a operação mais comumente invocada pelos sistemas de ficheiros.

O serviço de locking exporta uma API que permite aos clientes do serviço adquirir e libertar um trinco sobre um dado ficheiro através da disponibilização de duas operações: *tryAcquire(path, time)* e *release(path)*. A primeira operação permite ao cliente tentar reservar um ficheiro para escrita, sendo que retorna true em caso de sucesso, false caso contrário. A segunda operação liberta o trinco de um ficheiro, caso este esteja reservado. Na tabela 2 são apresentadas as operações feitas no DepSpace quando as operações *tryAcquire* e *release* são executadas.

Op. do Serviço de Locking	**Op. do DepSpace**
tryAquire(path, time)	cas(⟨LOCK, *, path⟩, ⟨LOCK, id, path⟩, time); **ou** renew(⟨LOCK, path⟩, time);
release(path)	inp(<LOCK, id, path>);

Tabela 2. Interacção do serviço de *locking* com o DepSpace

Um cliente só adquire o trinco para um ficheiro se não existir um tuplo com um trinco para o mesmo ficheiro no espaço (operação *cas*).

Para prevenir que outros clientes C2FS libertem trincos que não lhes pertencem, através da remoção dos tuplos que representam trincos, o serviço de locking configura o tuplo LOCK inserido no espaço com permissões de escrita apenas para o seu criador [4]. Desta forma, apenas quem cria o tuplo de LOCK pode removê-lo.

Para além disso, o uso de tuplos LOCK com tempos de expiração (que desaparecem após um intervalo de tempo), garantem que os trincos de ficheiros acabam por ser libertados se os seus detentores falharem. Note-se que o serviço de locking deixa para o cliente a responsabilidade de renovar este trinco, através de nova chamada da operação *tryAcquire(path, time)*, que aumenta o tempo de validade do tuplo (através operação *renew* do DepSpace).

5 Serviço de Armazenamento dos Dados

O serviço de armazenamento de dados do C2FS usa o DepSky modificado para gerir os dados em várias clouds. Assim como no serviço de directorias, este serviço dispõe de uma cache local que aumenta o desempenho do sistema de ficheiros e diminui os custos monetários associados a downloads que, desta forma, são evitados. Os dados deste serviço são armazenados dentro de contentores, cada um com um identificador único, utilizado pelos clientes do serviço para aceder e alterar os dados.

Este serviço implementa dois níveis de cache, cuja utilização pode ser definida pelo cliente do serviço, e que fazem variar as garantias fornecidas pelo sistema e o desempenho do mesmo. No primeiro nível de cache todas as escritas e leituras são feitas no disco local, enquanto que no segundo nível é utilizada a memória volátil para satisfazer os pedidos do cliente, sendo os dados mantidos neste nível enquanto o cliente assim o desejar. Embora o primeiro nível de cache diminua o desempenho do serviço, este fornece a garantia de persistência dos dados mesmo perante a falha da máquina onde o serviço de armazenamento está a correr. Em ambos os casos, apenas são utilizados os dados em cache caso não exista nenhuma versão mais recente destes armazenada nas clouds. Para invalidar dados

em cache, o serviço entrega ao cliente a responsabilidade de, aquando de uma operação sobre um contentor, fornecer o resumo criptográfico da versão mais recente desse contentor.

O serviço permite ao cliente fazer escritas (bloqueantes ou não) de contentores para a *cloud-of-clouds*, tendo esta escolha influência nas garantias de consistência dos dados em cache fornecidas pelo serviço. No caso de serem utilizadas escritas não-bloqueantes, para cada contentor a ser escrito, é lançada uma tarefa em *background* para enviar os dados. Neste caso, no retorno da operação, o serviço garante que os dados ficam armazenados no disco local, o que, embora diminua as garantias de consistência da cache, representa um aumento do desempenho das escritas. Por outro lado, no caso das escritas bloqueantes, o cliente é obrigado a esperar a confirmação da escrita do contentor nas clouds, tendo a garantia que, após a conclusão da operação, o contentor está correctamente armazenado. A utilização deste tipo de escritas fornece também tolerância à falha do disco local. Em ambos os casos, antes do envio dos dados para a *cloud-of-clouds*, os dados são cifrados pelo serviço utilizando uma chave simétrica fornecida pelo cliente.

Este serviço permite também ao cliente apagar um determinado contentor. Por questões de desempenho, o contentor é apagado da *cloud-of-clouds* através de uma tarefa que corre em *background*. Para tolerar a falha do serviço durante a execução de uma tarefa que apaga um contentor, é mantida em disco uma lista dos contentores a serem apagados em cada instante.

6 Agente C2FS

O agente C2FS é responsável por fazer com que as operações sobre ficheiros recebidas pela interface FUSE sejam executadas nos serviços de directorias, *locking* e armazenamento. Note-se que, aquando da montagem deste sistema de ficheiros por um utilizador, o agente C2FS cria uma instância de cada um dos serviços.

O C2FS controla tentativas de escrita concorrentes feitas sobre um determinado ficheiro por diferentes utilizadores. Para isto, o agente C2FS usa o serviço de *locking* para tentar adquirir o trinco na abertura de um ficheiro para escrita e liberta-lo no seu fecho, renovando entretanto a sua validade se necessário. Note-se no entanto que um ficheiro aberto para escrita pode ser aberto para leitura por um número ilimitado de clientes.

Em termos de consistência dos ficheiros, o C2FS satisfaz o modelo de consistência ao fechar [8], onde só é garantido que os dados escritos num ficheiro estejam visíveis a outros clientes aquando do fecho do ficheiro. Este modelo foi escolhido por duas razões básicas. Em primeiro lugar, porque ele combina bastante bem com o controlo de concorrência pessimista (com apenas um escritor por vez) empregado no sistema. Além disso, é um modelo bastante eficiente na medida em que tende a economizar o envio de escritas parciais para a cloud, favorecendo em seu lugar o envio de ficheiros consolidados.

O C2FS faz também controlo de acesso aos ficheiros, nos mesmos moldes do POSIX, utilizando o mecanismo de controlo de acesso de tuplos do DepSpace [4]. Como o serviço de directorias manipula nós armazenados como tuplos no DepSpace, um cliente só consegue aceder aos metadados de um ficheiro (e a partir daí obter

a chave para decifrar os dados do serviço de armazenamento) se tiver permissões de leitura sobre o tuplo correspondente ao nó. A abordagem utilizada consiste em guardar a chave simétrica usada pelo serviço de armazenamento para cifrar os dados de um ficheiro juntamente com os metadados do mesmo. Desta forma, um cliente que não tem permissões de leitura sobre um ficheiro, não consegue obter os metadados deste e, desta forma, não consegue obter a chave para decifrar os dados obtidos da *cloud-of-clouds*. Como se pode perceber, a chave usada para cifrar/decifrar os dados tem de ser alterada quando existe uma alteração das permissões de leitura de um determinado utilizador sobre um ficheiro. Caso isto não fosse feito, e visto o serviço de armazenamento não fazer controlo de acesso aos contentores de dados, o utilizador ao qual foram retiradas as permissões de leitura sobre ficheiro poderia obter os dados desse ficheiro através do serviço de armazenamento e decifrar o seu conteúdo. Para garantir que isto não acontece, o agente C2FS da máquina do onde é feita a alteração das permissões, reescreve os dados cifrados utilizando uma nova chave e, em seguida, actualiza os metadados do ficheiro com essa chave.

Para invalidar a cache do serviço de armazenamento, sempre que o agente C2FS lê ou escreve dados de um ficheiro, o resumo criptográfico destes dados é guardado nos metadados do ficheiro. Assim, a cada nova leitura dos dados, o agente C2FS fornece o resumo obtido do serviço de directorias (que pode ter sido actualizado por um outro utilizador) ao serviço de armazenamento que compara este resumo com o resumo dos dados que estão em cache. Caso estes não coincidam, a nova versão dos dados é obtida através do DepSky.

Para apagar um determinado ficheiro, o agente C2FS primeiro apaga os metadados do ficheiro e, só em seguida apaga o seu contentor. Esta abordagem permite ao serviço de armazenamento apagar os contentores em *background*, pois, após apagar os metadados do ficheiro, este deixa de ser visível aos outros utilizadores do sistema de ficheiros.

O C2FS permite ao utilizador configurar as garantias de consistência e durabilidade que lhe são fornecidas. O sistema de ficheiros garante que, aquando do fecho de um ficheiro (operação *close*), os dados em cache serão enviados para a *cloud-of-clouds* e que após a chamada da operação *fsync* os dados são enviados, ou para o disco local da máquina, ou para a *cloud-of-clouds*. É então possível ao utilizador, aquando da montagem do sistema de ficheiros, definir se as operações *close* e *fsync* devem ou não ser bloqueantes; se a operação *fsync* força a transferência os dados em cache para o disco ou para as clouds (usando o DepSky); e ainda permite ao utilizador definir o valor de Δ, usado pelo serviço de directorias.

7 Avaliação

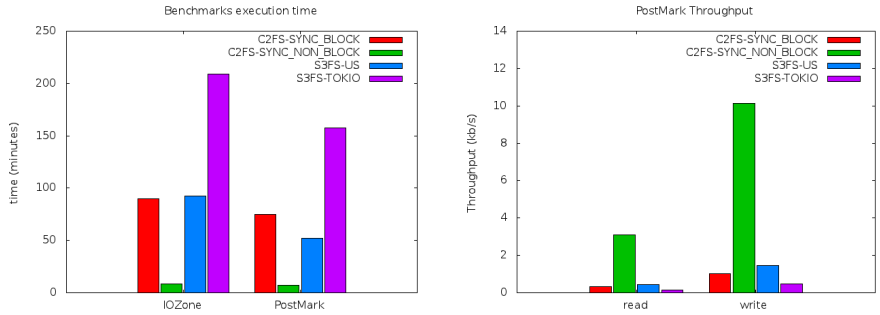
Nesta secção apresentamos uma avaliação preliminar do C2FS utilizando dois benchmarks diferentes: IOzone [2] e Postmark [10]. Todos os valores apresentados são os melhores resultados obtidos de cerca de 10 execuções.

O IOzone é um microbenchmark que executa operações de I/O no sistema de ficheiros. Este benchmark foi configurado para executar operações de I/O em

ficheiros com um máximo de 4MB. Já o Postmark tem por objectivo testar o serviço de directorias do sistema, simulando o tipo de operações que um servidor de email faria.

Para fins de comparação, os benchmarks foram executados no C2FS e no S3FS (um sistema de ficheiros para clouds, que armazena os dados e metadados utilizando o Amazon S3). Estes testes foram executados sobre o S3FS considerando armazenamento nas zonas de disponibilidade dos Estados Unidos e Tóquio[4], de forma a percebermos até que ponto a zona de disponibilidade condiciona o desempenho dos sistemas de ficheiros para clouds.

Quanto às configurações do C2FS, o DepSky foi configurado para utilizar quatro diferentes zonas de disponibilidade do Amazon S3 para armazenar os seus dados (simulando uma *cloud-of-clouds*), sendo elas UE (Irlanda), Oeste dos EUA (Norte da Califórnia), EUA padrão e Ásia-Pacífico (Tóquio). O DepSpace foi instalado em quatro máquinas virtuais Linux na Amazon EC2, localizadas na Irlanda, com 4 processadores (dois núcleos cada) e 7.5 GB de memória. Todos os testes foram efectuados com o cliente dos vários sistemas localizado em Portugal (Lisboa), num computador com 1 processador (2 núcleos a 2.4 GHz cada) e com 4 GB de memória.



(a) Tempo de execução. (b) Débito (em MB/s) no Postmark.

Fig. 2. Avaliação de desempenho do C2FS e comparação com o S3FS.

Nos gráficos da figura 2 mostramos o desempenho dos sistemas de ficheiros C2FS e S3FS quando testados com os benchmarks referidos acima. Como já referido, o C2FS permite que os clientes escolham entre vários modos de operação que garantem diferentes garantias de consistência entre os dados em cache e os guardados nas clouds. Assim, na figura 2 mostramos resultados do C2FS com diferentes garantias de consistência, assim como o desempenho do S3FS nas zonas de disponibilidade já referidas anteriormente.

Na figura 2(a) mostramos os tempos de execução de cada benchmark. Como podemos verificar, o nosso pior tempo, aquando da configuração do C2FS para efectuar escritas bloqueantes nas operações *close* e *fsync* (C2FS-SYNC_BLOCK),

[4] A escolha justifica-se pois, segundo as nossas medições, os Estados Unidos têm o melhor tempo de resposta, e Tóquio tem o terceiro melhor (o DepSky necessita de três respostas para obter os dados).

assemelha-se ao melhor tempo para o S3FS, quando este armazena os seus dados nos Estados Unidos (S3FS-US). Os tempos mais elevados para cada benchmark foram medidos quando o S3FS utiliza a região de Tóquio para armazenar os seus dados (S3FS-TOKIO). Note que Tóquio também é uma das localizações utilizadas pelo C2FS. Os melhores tempos medidos foram obtidos na utilização do C2FS com o envio dos dados em *background* (C2FS-SYNC_NON_BLOCK).

Na figura 2(b) mostramos a taxa de transferência máxima obtida nos testes do Postmark. Aqui, tal como na anterior comparação, o melhor desempenho pertence ao C2FS com escritas não bloqueantes, mostrando uma taxa de transferência de 3 KB/s nas leituras e de 10 KB/s nas escritas. Mais uma vez, a utilização do C2FS com sincronização bloqueante tem um desempenho comparável ao do S3FS configurado para armazenar os seus dados nos Estados Unidos. O S3FS com armazenamento em Tóquio volta a obter os piores resultados dos casos estudados (0,15 KB/s para as leituras, 0,48 KB/s para as escritas). Os resultados mostram que o C2FS com sincronização bloqueante consegue uma performance bastante aceitável quando comparado ao S3FS a enviar dados para os Estados Unidos padrão tendo em conta a diferença de garantias fornecidas. O C2FS com sincronização não-bloqueante mostra um desempenho muito superior a qualquer um dos outros estudados, oferecendo aos utilizadores deste sistema a possibilidade de abdicar de algumas garantias de consistência, em troca de um aumento bastante razoável do desempenho do sistema.

8 Trabalhos Relacionados

Hoje em dia o conceito de armazenar os dados em clouds é utilizado por alguns sistemas de ficheiros [3,11,13]. A maioria destes sistemas baseiam-se no modelo cliente-servidor desenvolvido por alguns sistemas de ficheiros tradicionais [8,12]. O que diferencia estes novos sistemas de ficheiros é a sua performance e desempenho tendo em conta as garantias que oferecem. Uma das suas principais limitações consiste em confiarem num único provedor de armazenamento para armazenar os dados, dependendo a disponibilidade dos dados do sistema da disponibilidade deste provedor.

O C2FS vem responder a esta necessidade fornecendo uma interface de sistema de ficheiros para a utilização das garantias providas pelo DepSky [5], que armazena os dados numa *cloud-of-clouds*. O C2FS dispõe de um serviço de directorias distribuído que recorre a um serviço de coordenação, o DepSpace [4], para armazenar os metadados do sistema dando garantias de confidencialidade, disponibilidade e tolerância a faltas bizantinas. Tal como este, também o serviço de directorias do Farsite [7] é distribuído, não fornecendo, no entanto, tolerância a faltas bizantinas nem recorrendo a uma abstracção genérica fornecida por um serviço de coordenação para guardar os metadados.

9 Conclusões

Neste artigo apresentamos a arquitectura e avaliação preliminar do C2FS, um sistema de ficheiros multi-utilizador, que facilita o armazenamento de dados em várias clouds, e ainda tolera falhas por parte dos clientes e provedores.

Este sistema fornece ainda disponibilidade, integridade e confidencialidade, tanto dos dados como dos metadados, através da utilização de sistemas tolerantes a intrusões como o DepSpace e o DepSky [4,5].

Como os resultados mostram, o C2FS tem um desempenho aceitável quando comparado outros sistemas de ficheiros para armazenamento em clouds (e.g., S3FS). Ainda assim, existem algumas melhorias a efectuar de modo a melhorar a performance do sistema de ficheiros e corrigir limitações que este apresenta.

Agradecimentos. Agradecimentos especiais a João Sousa, João Felix e Marcel Santos pelo apoio com o DepSpace. Este trabalho foi suportado pela Comissão Europeia através do projecto TClouds (FP7/2007-2013, ICT-257243) e pela FCT através de seu programa multianual (LaSIGE) e do projecto CloudFIT (PTDC/EIA-CCO/108299/2008).

Referências

1. FUSE: File System in Userspace. http://fuse.sourceforge.net/.
2. IOzone. http://www.iozone.org/.
3. S3FS: A file system backed by Amazon S3. http://code.google.com/p/s3fs/.
4. A. N. Bessani, E. P. Alchieri, M. Correia, and J. S. Fraga. DepSpace: a Byzantine fault-tolerant coordination service. In *Proc. of the 3rd ACM European Systems Conference – EuroSys'08*, pages 163–176, Apr. 2008.
5. A. N. Bessani, M. Correia, B. Quaresma, F. Andre, and P. Sousa. DepSky: Dependable and Secure Storage in cloud-of-clouds. In *Proc. of the 3rd ACM European Systems Conference – EuroSys'11*, Apr. 2011.
6. M. Burrows. The Chubby lock service. In *Proceedings of 7th Symposium on Operating Systems Design and Implementation – OSDI 2006*, Nov. 2006.
7. J. R. Douceur and J. Howell. Distributed directory service in farsite file system. In *Proc. of the 7th Symposium on Operating Systems Design and Implementation – OSDI 2006*, Nov. 2006.
8. J. H. Howard, M. L. Kazar, M. S. G., D. N. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. In *ACM Trans. Comput. Syst.* vol. 6, no. I, February 1988.
9. P. Hunt, M. Konar, F. Junqueira, and B. Reed. ZooKeeper: Wait-free coordination for Internet-scale services. In *Proc. of the USENIX Annual Technical Conference – USENIX 2010*, June 2010.
10. J. Katcher. PostMark: A new file system benchmark. Technical report, Aug. 1997.
11. K. P. N. Puttaswamy, T. Nandagopal, and M. Kodialam. Frugal Storage for Cloud File Systems. In *Proc. of the 3rd ACM European Systems Conference – EuroSys'12*, Apr. 2012.
12. S. Shepler et al. Network File System (NFS) version 4 Protocol (RFC 3530). IETF Request For Comments, Apr. 2003.
13. M. Vrable, S. Savage, and G. M. Voelker. BlueSky: A cloud-backed file system for the enterprise. In *Proc. of the 10th USENIX Conference on File and Storage Technologies – FAST'12*, 2012.
14. S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proc. of the 7th Symposium on Operating Systems Design and Implementation – OSDI 2006*, Nov. 2006.