Melhorando a Fiabilidade e Segurança do Armazenamento em *Clouds*

Bruno Quaresma, Alysson Bessani, and Paulo Sousa

Laboratório de Sistemas Informáticos de Grande Escala,
Faculdade de Ciências da Universidade de Lisboa,
Lisboa, Portugal

Abstract. With the increasing popularity of *cloud* storage services, companies that deal with critical data start thinking of using these services to store medical records databases, historical data of critical infrastructures, financial data, among others. However, many people believe that information stored that way is vulnerable, despite the guarantees given by providers, which makes reliability and security the major concerns about *cloud* storaging. In this work we present DEPSKY, a system that improves the availability, integrity and confidentiality of information stored in the *cloud*.

Resumo. Com a crescente popularidade das *clouds* de armazenamento, empresas que lidam com dados críticos começam a pensar em usar estes serviços para armazenar bases de dados de registos médicos, históricos de infra-estruturas críticas, dados financeiros, entre outros. No entanto, muitas pessoas acreditam que informação armazenada num sistema deste tipo é vulnerável, apesar de todas as garantias dadas pelos fornecedores, o que faz da fiabilidade e da segurança as maiores preocupações sobre o armazenamento em *clouds*. Neste trabalho apresentamos o DEPSKY, um sistema que melhora a disponibilidade, integridade e confidencialidade de informação armazenada na *cloud*.

1 Introdução

Actualmente, muitas organizações começam a optar de forma progressiva pelo uso de *clouds* de armazenamento. Exemplos recentes são serviços como o Twitter e o Facebook que até há bem pouco tempo tinham os seus próprios *data centers* de armazenamento e hoje terceirizam parte deste serviço para a Amazon e o seu Simple Storage Service (Amazon S3) [1]. Esta tendência pode ser definida como o armazenamento de informação num sistema de armazenamento remoto mantido por terceiros. A Internet fornece a ligação entre o computador e esse sistema.

O armazenamento em *clouds* tem algumas vantagens sobre o armazenamento tradicional. Por exemplo, se se armazenar informação numa *cloud*, esta estará acessível a partir de qualquer local com acesso à Internet e evita a necessidade da manutenção de uma infra-estrutura de armazenamento (e.g., uma rede de discos) na organização. Além disso, o modelo de cobrança das *clouds* de armazenamento incorpora o conceito de elasticidade de recursos: paga-se apenas pelo uso e o serviço pode crescer arbitrariamente para tolerar altos picos de carga esporádicos.

À medida que as *clouds* de armazenamento se tornam mais e mais populares, empresas que lidam com dados críticos começam a pensar em usar estes serviços para

INForum 2010 - II Simpósio de Informática, Luís S. Barbosa, Miguel P. Correia (eds), 9-10 Setembro, 2010, pp. 625–636

armazenar bases de dados de registos médicos, históricos de infra-estruturas críticas, dados financeiros, entre outros. No entanto, um perigo muitas vezes ignorado está no facto dos sistemas de armazenamento remoto estarem fora do controlo dos donos dos dados, apesar das garantias dadas pelos fornecedores (e.g., SLAs de serviço), o que faz da fiabilidade e da segurança as maiores preocupações sobre o armazenamento em *clouds*. Neste trabalho apresentamos o DEPSKY, um sistema que garante disponibilidade, integridade e confidencialidade de informação armazenada na *cloud*. A ideia fundamental deste sistema é replicar a informação por várias *clouds* de armazenamento, utilizando algoritmos para armazenamento fiável e partilha de segredo. Além disso, apresentamos resultados experimentais que demonstram a viabilidade económica (especialmente para cargas de trabalho com bastantes mais leituras que escritas) e os benefícios em termos de desempenho e disponibilidade do sistema.

2 Clouds de Armazenamento

Existem diversos sistemas de armazenamento em *clouds*, uns possuem um foco muito específico, como armazenar apenas mensagens de e-mail ou imagens digitais, outros podem armazenar todo o tipo de dados. O seu funcionamento pode ser descrito da seguinte forma: um cliente envia ficheiros através da Internet para os servidores que guardam a informação. O acesso aos servidores pelo cliente é efectuado através de interfaces web ou serviços web (usualmente baseados no modelo REST - *REpresentational State Transfer*), que permitem o acesso e manipulação dos dados armazenados.

Nem todos os clientes estão preocupados apenas com a falta de espaço, alguns usam sistemas de armazenamento em *clouds* para *backup* de informação, o que garante que, caso haja algum problema na infra-estrutura computacional do cliente, a informação estará intacta na *cloud* de armazenamento.

Existem fornecedores de armazenamento em *clouds* que cobram uma quantia fixa por uma quota de espaço e largura de banda de entrada e saída de dados (ex., DivShare [3], DocStoc [4] e Box.net [2]), enquanto outros usam um modelo *pay-per-use* e cobram quantias variáveis consoante o espaço ocupado e a largura de banda utilizada pelo cliente (ex., Amazon S3 [1], Nirvanix [7], Windows Azure [6] e RackSpace [8]). Em geral, o preço do armazenamento *online* tem vindo baixar devido à entrada de cada vez mais empresas neste negócio.

As duas maiores preocupações acerca do armazenamento em *clouds* são a fiabilidade e a segurança. É improvável que uma organização confie os seus dados críticos a outra entidade sem a garantia que terá acesso a estes dados sempre que quiser (disponibilidade), que estes não serão corrompidos (integridade) e que mais ninguém terá acesso a eles sem a sua autorização (confidencialidade). Para garantir a segurança da informação, a maioria dos sistemas usa uma combinação de técnicas, incluindo:

- *Criptografia*: algoritmos criptográficos são usados para codificar a informação tornando-a ininteligível e quase impossível de decifrar sem a chave usada para cifrar a informação, normalmente uma chave secreta partilhada entre cliente e o serviço;
- *Autenticação*: é necessário o registo de um cliente através da criação de credenciais de acesso (ex., *username* e *password*);
- *Autorização*: o cliente define quem pode aceder à sua informação.

Mesmo com estas medidas de protecção, muitas pessoas acreditam que a informação armazenada num sistema de armazenamento remoto é vulnerável. Existe sempre a possibilidade de um *hacker* malicioso, de alguma maneira, ganhar acesso à informação do sistema, por exemplo, devido a vulnerabilidades existentes neste. Além disso, há sempre a preocupação de colocar os dados críticos (e muitas vezes confidenciais) nas mãos de terceiros, que terão acesso às informações neles contidos.

Finalmente, há também a questão da fiabilidade e disponibilidade dos serviços de armazenamento. Armazenar informação num sistema remoto acedido via Internet coloca a organização vulnerável a todos os problemas de conectividade e indisponibilidade temporária da Internet. Além disso, praticamente todos os grandes fornecedores de serviços de armazenamento já sofreram problemas de disponibilidade e/ou corromperam dados de clientes, mesmo com a redundância interna de seus sistemas (os dados são tipicamente armazenados em diferentes *data centers* do fornecedor).

3 DEPSKY

Nesta secção é apresentado o DEPSKY, um sistema para replicação de dados que melhora a fiabilidade e segurança da informação armazenada em *clouds*.

Os blocos atómicos de dados no DEPSKY designam-se por *unidades de dados* (*data units*), que podem ser actualizadas pelos seus donos e acedidas por um conjunto arbitrário de leitores. A disponibilidade destas unidades é garantida mesmo em caso de falhas devido ao uso de algoritmos de replicação para sistemas de quóruns bizantinos de disseminação [14], onde os dados armazenados em cada servidor (i.e., que neste caso são *clouds* de armazenamentos) são auto-verificáveis graças ao uso de assinaturas digitais e resumos criptográficos (i.e., se um servidor alterar o conteúdo dos dados, o leitor descobre e ignora os dados corrompidos).

O DEPSKY oferece também a possibilidade da informação mais sensível ser protegida através de um esquema de partilha de segredos [16], introduzindo garantias de confidencialidade: nenhuma *cloud*, individualmente, tem acesso à informação contida nos dados.

3.1 Modelo de Sistema

O modelo de sistema utilizado no DEPSKY segue uma série de hipóteses pragmáticas tidas em conta no desenho dos protocolos de replicação em *clouds*.

Cada *cloud* é representada por um servidor passivo (não executa nenhum código dos protocolos) que oferece operações de leitura e escrita de dados com semântica de consistência regular [12]: uma operação de leitura executada concorrentemente com uma operação de escrita retorna o valor da unidade de dados antes da escrita ou o valor que está a ser escrito.

Assumimos também que para cada unidade de dados há *apenas um escritor*, e este escritor só sofre *falhas por paragem*. Isto significa que cada bloco de dados é escrito por uma única entidade[1], o que simplifica os protocolos (que não têm de lidar com escritas concorrentes). Além disso, escritores maliciosos não são considerados pois estes

[1] Na prática pode existir mais de um escritor para uma unidade de dados desde que os acessos de escrita sejam feitos isoladamente (o que pode requerer algum controlo de concorrência).

poderiam escrever dados sem sentido do ponto de vista da aplicação de qualquer forma. Finalmente, estas duas hipóteses permitem a concretização de protocolos de leitura e escrita em sistemas onde os servidores são apenas discos passivos, como as *clouds* de armazenamento.

Os servidores (*clouds*) e leitores estão sujeitos a faltas arbitrárias ou bizantinas [13]. Da mesma forma, como são suportados múltiplos leitores para cada unidade de dados, é conveniente também assumir que estes podem ter qualquer comportamento. Devido ao uso de sistemas de quóruns bizantinos de disseminação [14], o sistema requer $n \geq 3f + 1$ servidores para tolerar até f servidores faltosos. O uso de sistemas de quóruns bizantinos também permite que o sistema tolere um número ilimitado de leitores faltosos.

Finalmente, assumimos a ausência de um sistema de distribuição de chaves entre os clientes. Os leitores apenas sabem como aceder ao sistema para ler dados e para isso possuem a chave pública do escritor para verificação e validação de dados.

3.2 Modelo de Dados

A figura 1 apresenta o modelo de dados do DEPSKY em três níveis. Num nível conceptual temos os blocos representados por *unidades de dados* (*data units*) que contêm, além do seu valor, um número de versão e informações de verificação que tornam os dados auto-verificáveis. Genericamente, uma unidade de dados do DEPSKY é representada em cada *cloud* por dois ficheiros: um contendo os metadados e o outro com o valor mais recente armazenando na unidade. Estes dois ficheiros estão sempre dentro de um *container*. O *container* de uma unidade de dados, para além de conter os metadados e o valor actual, pode conter também versões anteriores do valor desta unidade. Cada unidade de dados tem um nome único que é o seu identificador. Este é usado para obter referências para o *container* e metadados dessas unidades nos protocolos definidos.

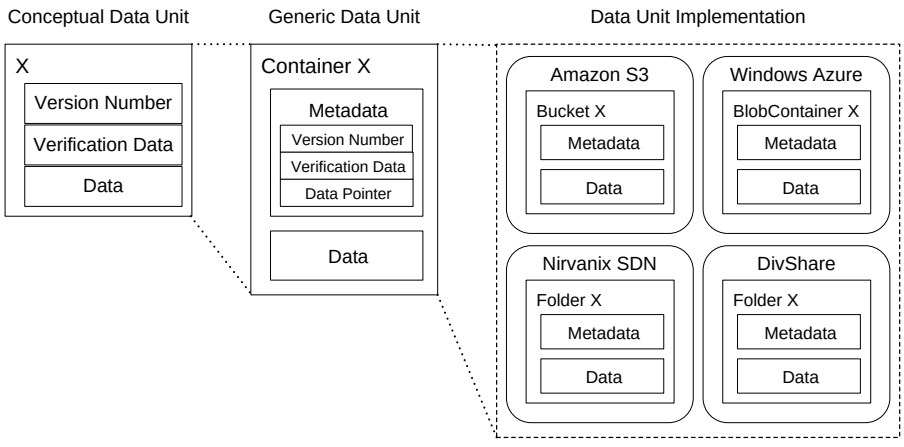


Figura 1. Decomposição do Data Unit X do DEPSKY, do conceito à concretização.

Os ficheiros de metadados são os mais importantes pois é sempre necessário um quórum destes nos protocolos definidos. Os metadados consistem na seguinte informação: um número de versão (*Version Number*), uma referência para o ficheiro com o valor desta versão (*Data Pointer*) e informação de verificação (*Verification Data*), que inclui um resumo criptográfico do valor para verificação de integridade deste e, no caso de ser uma unidade de dados com confidencialidade, dados públicos necessários para a leitura do valor. Para escrever ou ler uma unidade de dados é sempre necessário efectuar o *download* do ficheiro de metadados associado a esta em primeiro lugar.

3.3 ADS - *Available* DEPSKY

Esta secção apresenta o algoritmo ADS que promove uma melhoria da disponibilidade de dados na *cloud* através da replicação das unidades de dados por várias *clouds* de armazenamento.

Algoritmo de escrita.

1. Um cliente escritor começa por enviar um pedido de leitura dos metadados a todas as *clouds*. O escritor espera $n - f$ ficheiros de metadados correctamente assinados por ele e lidos de diferentes *clouds* para então obter o número de versão máximo dentre os contidos nestes ficheiros.
2. O número de versão lido no passo anterior é incrementado em uma unidade, dando origem ao número de versão dos dados a serem escritos nesta operação. Um ficheiro a conter os dados a serem escritos e cujo nome corresponde ao nome da unidade de dados concatenado com o número de versão é criado em todas as *clouds*. O escritor espera confirmação da escrita deste ficheiro de $n - f$ *clouds*.
3. Após conclusão da escrita da nova versão, são actualizados os metadados para a nova versão sendo enviados pedidos de escrita para este efeito. Neste passo o ficheiro de metadados é actualizado (ficheiro com metadados anterior é sobrescrito), ao contrário do passo 2 em que é escrita uma nova versão dos dados num ficheiro diferente do da versão anterior. A operação de escrita termina quando se recebe confirmação da actualização de metadados de $n - f$ *clouds*.

É importante referir que o algoritmo de escrita preserva as versões anteriores da unidade de dados. Estas versões podem ser apagadas quando o escritor achar conveniente através de um procedimento de *garbage collection* que envia pedidos de remoção suportados por todas as *clouds* estudadas.

Algoritmo de leitura.

1. Um cliente leitor começa por efectuar pedidos pelos metadados a todas as *clouds* e esperar por $n - f$ ficheiros de metadados correctamente assinados pelo escritor. O leitor obtém o número de versão máximo reportado nestes ficheiros.
2. Após obter o número de versão mais actual da unidade de dados, o cliente envia pedidos de leitura para esta versão a todas as *clouds* e espera a recepção de um valor cujo seu resumo criptográfico seja igual ao resumo criptográfico contido nos metadados, sendo então a operação terminada e este valor retornado.

Optimização de leitura. Uma optimização importante para diminuir os custos monetários do protocolo de leitura (ver secção 5.1) é enviar o pedido de leitura da versão mais actual do valor da unidade de dados apenas à *cloud* que responde mais rapidamente à requisição de metadados e reportar a versão mais actual dos dados. Desta forma, no melhor caso (sem falhas), apenas uma das *clouds* será lida. Caso esta *cloud* não responda atempadamente, outras *clouds* são acedidas até que se obtenha a informação desejada.

3.4 CADS - *Confidential & Available* DEPSKY

O ADS garante a integridade e disponibilidade dos dados em *clouds* de armazenamento. No entanto, um dos problemas fundamentais neste tipo de solução é evitar que entidades não autorizadas tenham acesso aos dados armazenados na *cloud*.

Esta secção apresenta o algoritmo CADS, que integra um algoritmo criptográfico de *partilha de segredos* de tal forma que os dados armazenados em cada *cloud* individualmente sejam de pouca utilidade para um terceiro que intercepte ou obtenha a informação.

Um esquema de partilha de segredos [16] é o método para dividir um segredo entre um grupo de n participantes, em que a cada um deles é atribuída uma parte do segredo (que tem o mesmo tamanho do segredo original). O segredo pode ser reconstruído apenas quando $f+1$ dessas partes são recombinadas e qualquer combinação de até f partes individuais não revelam nenhuma informação sobre o segredo.

A diferença fundamental entre os protocolos de escrita do ADS e do CADS é que neste último introduzimos o algoritmo de partilha de segredos no passo 2 do ADS, de tal forma a produzir tantas partes do segredo (valor a ser escrito na unidade de dados) quanto o número de *clouds*. Cada uma destas partes é depois enviada para a sua respectiva *cloud*.

O algoritmo de leitura do CADS funciona de forma bastante similar ao ADS, porém, ao invés de aguardar apenas uma resposta com a versão mais actual dos dados (ADS - passo 2), esperam-se $f + 1$ partes de diferentes *clouds* para combiná-las usando o algoritmo de partilha de segredos, obtendo assim o valor originalmente escrito.

4 Concretização

O DepSky e todos os seus componentes foram concretizados na linguagem de programação Java. Em primeiro lugar foram concretizados alguns controladores, que são responsáveis pela comunicação com os diferentes sistemas de armazenamento em *clouds*. Cada controlador comunica com a respectiva *cloud* através de seus serviços web disponibilizados, utilizando uma interface REST. Toda a comunicação é efectuada sobre HTTP (ADS) ou HTTPS (CADS[2]). Os controladores foram os componentes que mais tempo consumiram em termos de concretização dada a variedade no funcionamento dos diferentes serviços web de cada *cloud*.

Foram concretizados controladores (com seus respectivos números de linhas de código) para os seguintes serviços: Amazon Simple Storage Service (175 LOC), Microsoft Windows Azure Platform (200 LOC), Nirvanix Storage Delivery Network (280 LOC),

[2] Requer canais confidenciais para garantir que as partes do segredo gerado são obtidas apenas pelas *clouds* a que se destinam.

DivShare (350 LOC), DocStoc (350 LOC) e Box.net (380 LOC). Os controladores do Amazon S3 e do Windows Azure foram concretizados sobre bibliotecas fornecidas pelos fornecedores do serviço ligeiramente modificadas para suportarem *proxies*. Após a conclusão de um número suficiente de controladores iniciou-se o desenvolvimento do componente responsável pelos protocolos (600 LOC), e de outro responsável pela verificação, validação e criação de metadados do sistema (250 LOC). Foi também concretizado um *wrapper* para controladores que efectua a gestão dos *retries* e *timeouts* dos pedidos HTTP (150 LOC), para garantir fiabilidade fim-a-fim. Finalmente, utilizou-se a biblioteca JSS (*Java Secret Sharing*) [5] para concretizar o esquema de partilha de segredos.

5 Avaliação

Nesta secção apresentamos uma avaliação do DepSky que tenta responder a três perguntas: Qual o custo adicional da utilização de replicação em *clouds* de armazenamento? Qual o ganho de desempenho e de disponibilidade na utilização de *clouds* replicadas para armazenar dados? Qual o custo relativo das diferentes versões (ADS, ADS com leitura optimizada e CADS) do DEPSKY?

5.1 Custo do Armazenamento Replicado

As *clouds* de armazenamento usualmente cobram pela quantidade de dados que entram, saem e ficam armazenados nos seus *data centers*. A tabela 1 apresenta os custos da utilização do modelo de *unidade de dados* apresentado neste artigo em diversas configurações do DEPSKY e em diferentes *clouds* individualmente[3]. A tabela mostra o custo em USD da realização de 10.000 operações de leitura e escrita para diferentes tamanhos de blocos de dados.

Operação	Tamanho	DEPSKY	DEPSKY opt. (melhor caso)	Amazon S3 (EU)	RackSpace	Windows Azure	Nirvanix
10k Leituras	100 kb	0,69	0,12	0,11	0,22	0,16	0,18
	1 Mb	6,54	1,02	1,01	2,20	1,51	1,80
	10 Mb	65,04	10,02	10,01	22,0	15,01	18,0
10k Escritas	100 kb	1,10	1,10	0,20	0,28	0,11	0,18
	1 Mb	4,84	4,84	1,10	0,80	1,01	1,80
	10 Mb	46,24	46,24	10,10	8,00	10,01	18,0

Tabela 1. Custo estimado, em USD, de 10.000 operações de leitura e escrita de dados com 100KB, 1MB e 10MB. É de salientar que os protocolos de leitura do DEPSKY efectuam 2 pedidos de leitura a cada *cloud*, e também, que os protocolos de escrita efectuam um pedido de leitura e 2 pedidos de escrita a cada *cloud*.

A coluna "DEPSKY" apresenta os custos do uso dos protocolos ADS e CADS propostos no artigo. É importante referir que o uso de confidencialidade (protocolo CADS)

[3] Nesta tabela apresentam-se os custos do RackSpace ao invés do Divshare (usado nas experiências de latência) devido ao facto do primeiro cobrar por uso, enquanto o segundo oferece apenas pacotes fixos.

não apresenta acréscimo representativo de custo uma vez que os seus metadados ocupam 500 bytes enquanto os metadados para unidades de dados sem confidencialidade ocupam cerca de 250 bytes.
A coluna "DEPSKY opt. (melhor caso)" apresenta os custos quando a optimização de leitura para o protocolo ADS é empregue. Neste caso, a política de escolha da *cloud* de leitura tem em conta não a que retornou os metadados mais rapidamente mas sim a que apresenta menor custo de leitura.

Custo de leitura. Este custo corresponde apenas ao custo de se ler os metadados da unidade de dados e os dados propriamente ditos. O custo de leitura do DEPSKY é similar à soma dos custos de leitura em cada uma das quatro *clouds* individualmente, enquanto que na versão optimizada temos o custo similar à Amazon S3, com um acréscimo de poucos cêntimos devido ao acesso dos metadados em todas as *clouds*.

Custo de escrita. O custo da escrita considera o custo de se ler os metadados, escrever uma nova versão destes e escrever a nova versão dos dados. Além disso, neste custo incluímos os custos de armazenamento dos dados, o que significa que estamos a considerar um sistema onde nenhuma versão de uma unidade de dados será apagada (i.e., escritas apenas criam novas versões). Conforme já referido, esta funcionalidade é importante para dados críticos na medida em que permite recuperar versões anteriores das unidades de dados armazenadas. No entanto, na prática esse custo pode ser amortizado apagando-se versões antigas.

Os resultados da tabela mostram que o custo apresentado para as versões do DEPSKY correspondem à soma dos custos de escrita nas *clouds*. Estes custos, assim como na leitura não-optimizada, advêem do modelo de replicação utilizado onde armazenamos o bloco de dados em todas as *clouds*. Se aplicássemos técnicas similares ao RAID nível 5 [15], esses custos cairiam pela metade, já que os dados armazenados em cada *cloud* teriam aproximadamente metade do tamanho da unidade de dados.

5.2 Desempenho e Disponibilidade

O DEPSKY foi desenhado tendo em conta cargas de trabalho em que leituras são muito mais frequentes que escritas, como é observado em praticamente todos os sistemas de armazenamento [10]. Assim, a nossa avaliação concentrou-se na latência das operações de leitura em diferentes configurações. No entanto, são reportados alguns valores de latência do protocolo de escrita no fim desta secção para fins de completude do estudo.

Metodologia. As medidas de latência de leitura foram obtidas através de uma aplicação que acede aos dados de 7 formas diferentes (configurações): às 4 *clouds* de armazenamento individualmente (Amazon S3, Windows Azure, Nirvanix e Divshare) e às 3 versões do protocolo de leitura do DEPSKY (ADS, ADS com leituras optimizadas e CADS). Todas as versões do DEPSKY usam as quatro *clouds* mencionadas para armazenar dados, e portanto toleram uma falha.

Foram medidos tempos de leitura para unidades de dados de três tamanhos: 100K, 1M e 10M bytes. A aplicação executou todas estas leituras periodicamente - de um em um minuto (10K e 1M) ou de cinco em cinco minutos (10M) - e armazenou os

tempos obtidos em ficheiros de log. O objectivo foi ler os dados através das diferentes configurações num intervalo de tempo o mais curto possível tentando minimizar as variações de desempenho da Internet.

As experiências foram realizadas entre 31 de Maio e 14 de Junho de 2010, com o cliente a executar numa máquina do Departamento de Informática da FCUL e a aceder às quatro *clouds* de armazenamento. Foram efectuadas 99.414 medidas de latência, correspondento a 14.202 medidas por cada uma das 7 configurações.

Em todos os testes do DEPSKY, o custo dos algoritmos de criptografia (assinatura, verificação e partilha de segredos) foi inferior a 20 ms, o que corresponde a menos de 2% da menor latência observada nos protocolos. Isto era esperado uma vez que a latência de comunicação da Internet e o processamento adicional para acesso de serviços web tende a dominar o tempo de execução de qualquer aplicação neste ambiente.

Latência de leitura. A figura 2 apresenta a função de distribuição cumulativa das latências medidas na leitura dos diversos tamanhos dos dados nas diversas *clouds* individualmente e utilizando as diferentes versões do DEPSKY. São apresentados resultados relativos ao acesso às *clouds* individualmente (figuras 2(a), 2(c) e 2(e)) e utilizando diferentes versões do DEPSKY (com os resultados da Amazon S3, para fins de comparação - figuras 2(b), 2(d) e 2(f)).

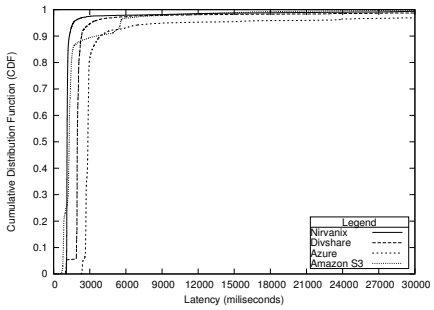
Nas unidades de dados de 100K podemos observar que as distribuições de latências para todas as configurações são bastante semelhantes. Já nas experiências com unidades de dados de 1M, podemos observar alguma discrepância entre os desempenhos da Amazon S3 e da Nirvanix. Além disso, com este tamanho de blocos já se percebe a diferença entre o uso de replicação de dados em *clouds* e uma única *cloud*: 90% das leituras optimizadas com o DEPSKY estão abaixo de 3,2 segundos, enquanto na S3 este valor aproxima-se dos 8 segundos.

As experiências com unidades de dados de 10M já demonstram as dificuldades em lidar-se com o armazenamento de dados na Internet. Na figura 2(e) observa-se uma larga discrepância entre os resultados observados para a Nirvanix e a Divshare quando comparados com os resultados da Azure e da S3. Em particular, no decorrer destas experiências observou-se um largo período de indisponibilidade da Azure (ver a seguir), o que é representado no gráfico pelos 20% dos resultados de latência que não aparecem na figura.

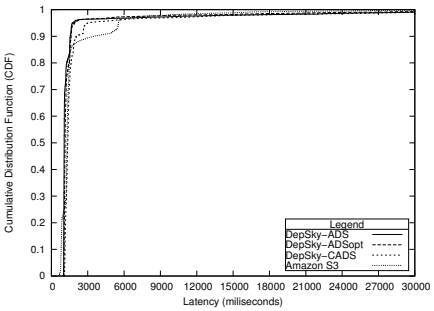
No que diz respeito às diversas versões do DEPSKY, a figura 2(f) mostra que neste caso a replicação dos dados em diversas *clouds* diminui de forma significativa a latência de leitura, mesmo na versão optimizada em que não se tenta ler de várias *clouds* mas apenas da que retornou os metadados mais rapidamente. De notar também que o uso da primitiva de partilha de segredos para confidencialidade (protocolo CADS), torna o DEPSKY muito menos eficiente já que é necessário obter os dados de duas *clouds* diferentes, ao invés de uma (como acontece nas outras versões).

Falhas. Durante as experiências foram observadas várias falhas no acesso aos sistemas de armazenamento, conforme reportado na tabela 2.

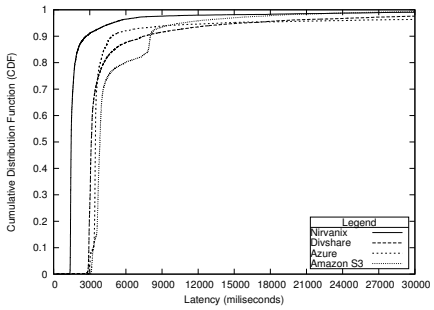
Durante os testes observámos um período de instabilidade e indisponibilidade na *cloud* Azure entre as 11h e as 21h do dia 10 de Junho (GMT+1, fuso horário de verão de Portugal continental). Neste período mais de 95% dos pedidos de leitura dos dados foram rejeitados com a mensagem de erro *"Unable to read complete data from server"*.



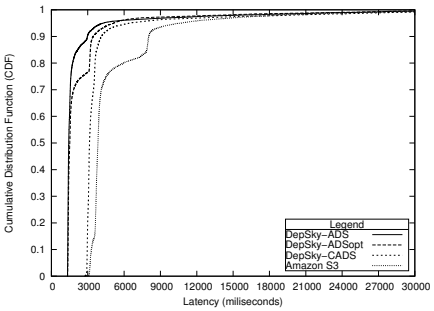
(a) Clouds - Unidades de dados de 100K.



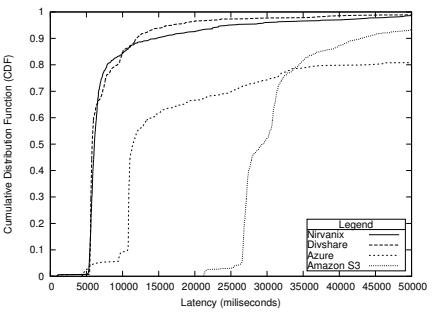
(b) DEPSKY - Unidades de dados de 100K.



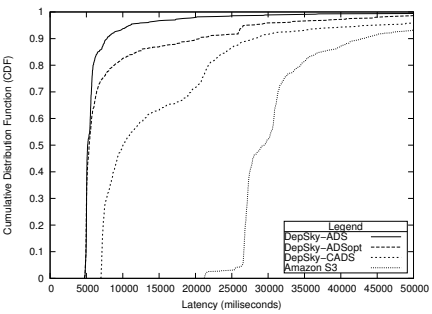
(c) Clouds - Unidades de dados de 1M.



(d) DEPSKY - Unidades de dados de 1M.



(e) Clouds - Unidades de dados de 10M.



(f) DEPSKY - Unidades de dados de 10M.

Figura 2. Função de distribuição cumulativa para as latências de leitura observadas em quatro diferentes *clouds* (Amazon S3, Windows Azure, Nirvanix e DivShare) e nas três versões do DEPSKY replicando dados nas mesmas *clouds*.

Experiência	Todas	Amazon S3	Azure	Nirvanix
100K	1	0	0	1
1M	1	9	13	2
10M	0	0	10+10hs	7

Tabela 2. Número de falhas observadas durante as experiências de leitura. O "10+10hs" para a Azure na unidade de dados de 10M significa que para além das 10 falhas reportadas houve um período de 10 horas onde mais de 95% dos acessos individuais a este sistema falharam.

Para além deste evento, o número de operações mal sucedidas é bastante pequeno se tivermos em conta a quantidade total de operações executadas. É de salientar que o DEPSKY falhou apenas duas vezes, quando todas as *clouds* estavam indisponíveis. Estas falhas aconteceram possivelmente devido a problemas de conectividade na saída da rede do DI/FCUL.

Latência de escrita. Para fins de completude da nossa avaliação reportamos na tabela 3 os tempos médios de escrita para diferentes configurações (obtidos a partir de um conjunto de 1000 escritas para cada tamanho de unidade de dados, executadas no dia 14 de Junho).

Dados	DEPSKY-ADS	DEPSKY-CADS	Amazon S3	DivShare	Azure	Nirvanix
100K	1767	2790	2336	3287	2801	2802
1M	4376	4928	5640	4684	3823	2626
10M	20315	24012	32204	8298	20017	4816

Tabela 3. Latência média (ms) de escrita para diferentes tamanhos de unidades de dados, configurações do DEPSKY e *clouds* de armazenamento.

Estes resultados mostram que algumas *clouds* (Divshare e Nirvanix) apresentam pouco aumento de latência quando passamos a escrever altos volumes de dados, enquanto outras (Azure e S3), apresentam uma perda de desempenho proporcional ao tamanho dos dados a serem escritos. O desempenho das versões do DEPSKY é similar a estas versões mais lentas na medida que os protocolos de escrita requerem a confirmação de escrita em 3 das 4 *clouds* utilizadas.

6 Trabalhos Relacionados

Até onde sabemos, existem apenas dois trabalhos bastante recentes que tentam fazer algo similar ao DEPSKY para melhorar a confiabilidade e segurança dos dados armazenados nas *clouds*, tendo sido ambos desenvolvidos em paralelo com o trabalho aqui reportado.

O HAIL (*High-Availability Integrity Layer*) [9] consiste num conjunto de protocolos criptográficos que juntam códigos de apagamento com provas de recuperação que permitem a concretização de uma camada de software para proteger a integridade dos dados armazenados em *clouds*, mesmo que estas sejam invadidas e corrompidas por um adversário móvel. Quando comparado ao DEPSKY, o HAIL apresenta pelo menos três limitações: só lida com dados estáticos (i.e., os algoritmos não suportam actualizações e múltiplas versões dos dados), requer que os servidores executem código (ao contrário do DEPSKY, que considera as *clouds* de armazenamento como discos passivos) e não usa nenhum mecanismo para protecção da confidencialidade dos dados armazenados.

O sistema RACS (*Redundant Array of Cloud Storage*) [11] utiliza técnicas similares às utilizadas nos sistemas RAID nível 5 [15] para concretizar replicação de dados em diversas *clouds*. Diferentemente do DEPSKY, o RACS não se preocupa com problemas de segurança, mas sim com possíveis "falhas económicas", onde uma *cloud* aumenta o custo do seu serviço de tal forma que torna inviável o acesso aos dados. Além de

não proteger contra corrupção de dados e violações de confidencialidade, o RACS também não suporta actualizações dos dados armazenados. Todas estas limitações tornam o RACS menos abrangente do que o DEPSKY.

Além das diferenças entre os sistemas, os trabalhos sobre o HAIL e RACS não apresentam nenhum tipo de medida que utiliza diversidade de *clouds*.

7 Conclusão

Neste trabalho foi apresentado o DEPSKY, um sistema que fornece disponibilidade, integridade e confidencialidade de informação armazenada na *cloud*. Na avaliação experimental demonstrou-se que o DEPSKY não fica muito aquém, em termos de desempenho, dos serviços testados. Podemos afirmar que com o DEPSKY temos sempre o melhor serviço independentemente das condições de cada *cloud*.

Os trabalhos actuais e futuros deverão concentrar-se na inclusão de códigos de apagamento para diminuir o tamanho dos blocos de dados armazenados (de forma similar ao RAID [15]) e numa avaliação da disponibilidade e desempenho das *clouds* a partir de diferentes localizações na Internet.

Agradecimentos. Este trabalho foi suportado pela FCT através de seu programa multianual (LaSIGE) e do projecto CloudFIT (PTDC/EIA-CCO/108299/2008).

Referências

1. Amazon Simple Storage Service (`https://s3.amazonaws.com`), June 2010.
2. Box.net (`http://www.box.net`), June 2010.
3. Divshare (`http://www.divshare.com`), June 2010.
4. Docstoc (`http://www.docstoc.com`), June 2010.
5. Java secret sharing (`http://www.navigators.di.fc.ul.pt/software/jitt/jss.html`), June 2010.
6. Microsoft Windows Azure Platform (`http://www.windowsazure.com`), June 2010.
7. Nirvanix Storage Delivery Network (`http://www.nirvanix.com`), June 2010.
8. Rackspace (`http://www.rackspace.com`), June 2010.
9. Kevin D. Bowers, Ari Juels, and Alina Oprea. HAIL: a high-availability and integrity layer for cloud storage. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, pages 187–198, New York, NY, USA, 2009. ACM.
10. Gregory Chockler, Rachid Guerraoui, Idit Keidar, and Marko Vukolić. Reliable Distributed Storage. *IEEE Computer*, 42(4):60–67, 2009.
11. L. Princehouse H. Abu-Libdeh and H. Weatherspoon. RACS: A case for cloud storage diversity. *ACM Symposium on Cloud Computing (SOCC)*, June 2010.
12. Leslie Lamport. On Interprocess Communication. Part I: Basic Formalism. *Distributed Computing*, 1(2):77–85, 1986.
13. Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.
14. Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, October 1998.
15. David A. Patterson, Garth Gibson, and Randy H. Katz. A case for redundant arrays of inexpensive disks (raid). In *SIGMOD '88: Proceedings of the 1988 ACM SIGMOD international conference on Management of data*, pages 109–116, New York, NY, USA, 1988. ACM.
16. Adi Shamir. How to Share a Secret. *Commun. ACM*, 22(11):612–613, 1979.