Armazenamento de Dados com Coerência Causal na Periferia da Rede

Nuno Afonso[1], Manuel Bravo[1,2] e Luís Rodrigues[1]

[1] INESC-ID, Instituto Superior Técnico, Universidade de Lisboa, Portugal
{nuno.c.afonso,ler}@tecnico.ulisboa.pt
[2] Université Catholique de Louvain, Bélgica
angel.bravo@uclouvain.be

Resumo Neste artigo apresentamos uma arquitetura para suportar o armazenamento de dados na periferia da rede com garantias de coerência causal. Manter a coerência causal na periferia acarreta novos desafios, devido a ser necessário lidar com um elevado número de réplicas, o facto das réplicas terem recursos limitados, a topologia da rede ser potencialmente muito dinâmica e a migração dos clientes entre diferentes réplicas ser demorada. A nossa solução recorre a estampilhas temporais multiparte, mas de tamanho constante, que permitem suportar elevado débito nas atualizações, oferece baixa latência na aplicação das mesmas, e suporta a mobilidade dos clientes de modo eficiente.

Keywords: Coerência Causal · Computação na Periferia.

1 Introdução

Atualmente, a computação na nuvem é um paradigma estabelecido e a maioria das aplicações que se executam em dispositivos móveis usam-na de diferentes formas: para aceder a dados armazenados remotamente, enviar informação ou delegar computações mais intensivas que possam reduzir a autonomia dos dispositivos [20]. O processamento de imagem para o reconhecimento de rostos ou de objetos [22] é um exemplo de uma tarefa que pode ser delegada na nuvem.

Apesar das vantagens que a utilização dos recursos da nuvem acarreta, esta aproximação possui também limitações fundamentais. Em particular, o acesso a um serviço que se executa na nuvem é afetado pela latência no acesso aos centros de dados remotos. Esta latência pode ser demasiado elevada para aplicações interativas de realidade virtual ou colaborativas. Estas aplicações necessitam de um tempo de resposta abaixo de $5-30$ milissegundos para serem utilizáveis [19].

Uma estratégia para reduzir a latência acima referida é recorrer a dispositivos que se encontram perto dos utilizadores (computação na periferia da rede). Existem diferentes variantes deste princípio, com designações distintas, tal como computação móvel na periferia (MEC) [18], utilização de micro centros de dados [21] (também designados por *cloudlets*, que traduzimos para *cúmulos*[1]) e

[1] Os cúmulos são um dos tipos de nuvens mais comuns, que tipicamente possuem uma base com baixa altitude.

computação realizada por uma malha de dispositivos de pequena capacidade que forma uma *neblina* (do inglês, *fog computing)*. Para além de reduzir a latência, a computação na periferia reduz a carga nos centro de dados colocados na nuvem [12] e aumenta a robustez em períodos de inacessibilidade aos mesmos [19]. No entanto, as máquinas na periferia têm menos recursos. O seu armazenamento guarda menos dados, forçando uma *replicação parcial*. A replicação parcial e o deslocamento físico dos utilizadores finais obrigam a que a mudança de réplica a que o cliente se liga seja feita com impacto mínimo.

Neste artigo, escolhemos o modelo de cúmulos, que executam operações a pedido dos clientes e que mantêm também cópias locais dos dados acedidos com mais frequência pelos utilizadores locais [4]. Apesar desta restrição, o sistema só necessita de pontos-de-presença que sejam controlados pelos provedores de serviços (comum a todos os modelos). A principal vantagem é a de este recorrer a máquinas com mais recursos [5], facilitando a transição desde a nuvem. As cópias locais permitem aos cúmulos responder à grande maioria dos pedidos, sem necessitar de aceder aos centro de dados colocados na nuvem [17]. Naturalmente, os cúmulos replicam parte da informação disponível na nuvem. Uma vez que os mesmos dados poderão ser replicados em vários cúmulos e nos centros de dados mantidos no resto da nuvem, é necessário assegurar que os clientes observam um estado coerente, independentemente do cúmulo a que se ligam. Esta tarefa é dificultada pela existência de um elevado número de cúmulos. De facto, para assegurar a latência alvo de $5-30$ milissegundos, estimamos que qualquer região fortemente habitada com 95 114 Km^2 (círculo com um raio de 174 Km) deva manter um cúmulo local. Este resultado foi obtido através dos tempos de ida e volta (RTT) entre regiões Amazon EC2 (o RTT é $22,5$ milissegundos entre Oregon e Califórnia) separadas por 786 Km, respeitando o limite mínimo de 5 milissegundos (distância percorrida em metade deste intervalo). Com uma área de $10,18$ milhões de Km^2, a Europa requer mais de 100 cúmulos.

Apresentamos uma nova arquitetura hierárquica que expande o armazenamento mantido na nuvem por múltiplos cúmulos que se executam na periferia da rede. Esta arquitetura foi desenhada para assegurar coerência causal no acesso aos dados. A nossa solução recorre a estampilhas temporais multi-parte, mas de tamanho constante, que permitem suportar elevado débito nas atualizações, baixa latência na aplicação das mesmas, e a mobilidade dos clientes de modo eficiente. As soluções atuais não satisfazem simultaneamente todos este requisitos: algumas são desenhadas para um reduzido número de réplicas [6, 9, 11, 15, 24], para replicação total [9,11,15] ou atrasam a mudança de réplica [7]. Nós construímos um protótipo que usa a arquitetura proposta, a partir do Saturn [7], uma solução para armazenamento com coerência causal na nuvem. A nossa avaliação mostra que a nossa arquitetura consegue lidar eficientemente com um elevado número de cúmulos, a replicação parcial, e a impossibilidade de manter topologias de disseminação complexas. O débito de operações suportadas, a latência na aplicação das atualizações, e a rapidez na migração de clientes entre diferentes cúmulos é significativamente melhor do que em soluções anteriores, considerando o elevado número de cúmulos.

2 Coerência Causal nos Cúmulos

Nesta secção, apresentamos o desenho de um sistema de armazenamento para a periferia da rede que oferece coerência causal. Começamos por uma panorâmica da sua arquitetura e das suas principais características. Posteriormente, descrevemos os vários protocolos que asseguram a coerência dos dados.

2.1 Panorâmica

Consideramos que a rede pode ser dividida em *regiões*, de forma em que cada região existe pelo menos um centro de dados na nuvem e um conjunto de cúmulos. O número de cúmulos em cada região depende da área da mesma e do número de clientes esperados, em que o objetivo é assegurar que os clientes conseguem aceder a um cúmulo não sobrecarregado, interagindo com baixa latência. Assumimos também que o centro de dados que reside na região possui uma cópia total dos dados que os clientes necessitam e que os cúmulos possuem apenas cópias parciais desses dados. Desta forma, cada item de dados possui uma réplica na nuvem e, provavelmente, em cúmulos (0 ou mais). Os mecanismos e protocolos propostos neste artigo são agnósticos em relação às políticas para população das réplicas. Uma possibilidade é catalogar a informação com geo-referenciação e manter cópias dos dados nos cúmulos mais próximos [23]; a otimização destas políticas é um tema ortogonal às contribuições deste artigo.

Propomos também usar uma separação entre a propagação do conteúdo das atualizações e a propagação da informação de controlo que as permite ordenar de forma a respeitar a causalidade. Sempre que é feita uma atualização de uma réplica, é gerada uma *etiqueta* que pode ser usada para ordenar essa atualização em relação a outras atualizações. O conteúdo das atualizações é propagado para todas as réplicas e, em paralelo, existe um sistema de propagação de etiquetas que entrega a cada réplica uma serialização destas etiquetas coerente com a causalidade. A serialização permite reduzir os metadados, tornando-se a melhor opção. O conteúdo das atualizações pode ser entregue às réplicas por qualquer ordem. No entanto, este conteúdo será tornado visível respeitando a ordenação imposta pelo serviço de distribuição de etiquetas. O serviço de propagação de etiquetas baseia-se numa topologia em estrela, onde o centro de dados e todos os cúmulos se encontram ligados a um *corretor de etiquetas regional*. Este corretor recebe as etiquetas geradas em cada réplica, realiza processamento sobre as mesmas, e distribui as etiquetas após este processamento para cada réplica. O processamento e re-encaminhamento de etiquetas é feito de forma a assegurar que as réplicas conseguem garantir aos clientes que estes só observam um estado coerente com a causalidade. Esta arquitetura está ilustrada na Figura 1.

2.2 Metadados

A grande maioria dos sistemas que oferecem garantias de coerência causal no acesso a dados partilhados baseiam-se na troca e processamento de metadados,

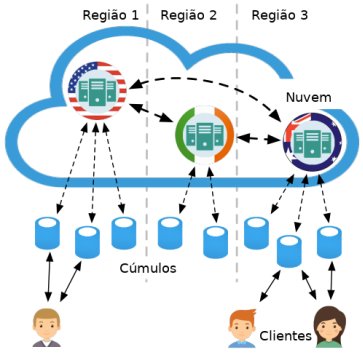

Figura 1. Arquitetura hierárquica do nosso sistema: as três camadas.

que capturam as relações causa-efeito criadas no sistema. Num sistema distribuído, os metadados podem ser propagados de várias formas e ter diferentes dimensões [6, 7, 9–11, 14–16, 24]. A grande maioria propaga os dados conjuntamente com os metadados [9–11, 14, 15, 24], mas há também aqueles que os separam [6, 7]. Estes últimos, apesar de aumentarem o tráfego de controlo, permitem uma arquitetura mais modelar. Como referimos anteriormente, neste trabalho desacoplamos a gestão dos dados e dos metadados.

A dimensão dos metadados está correlacionada com a precisão do rastreamento de dependências [8]; tipicamente um maior volume de metadados evita *falsas dependências* (as falsas dependências obrigam um cliente a aguardar por atualizações que não depende de facto, mas das quais parece depender devido a artefactos que resultam da gestão dos metadados). Infelizmente, o desempenho dos sistemas reduz-se com o aumento dos metadados [7, 16].

Para assegurar a coerência causal na periferia da rede, utilizamos os seguintes metadados, que oferecem um bom compromisso entre a sua dimensão (que permanece pequena e constante) e o impacto das falsas dependências que são geradas. A cada atualização é associada uma etiqueta com duas partes: uma *estampilha temporal local* (*etl*) que é gerada no cúmulo em que a atualização for realizada (e que permite ordenar a atualização em relação a outras atualizações feitas no mesmo cúmulo) e uma *estampilha temporal regional* (*etr*), que é gerada pelo corretor de metadados regional, e que permite ordenar a atualização em relação a atualizações gerados noutros cúmulos ou no centro de dados da nuvem. Ambas as estampilhas incluem vários campos, nomeadamente uma *origem*, *o*, que indica qual o componente do sistema onde a estampilha foi criada (um cúmulo, no caso das *etl*, e um corretor regional, no caso da *etr*), e um *relógio*, *r*, que captura o instante de criação (*físico* para a *etl* e *lógico* para a *etr*). O par de relógios forma um *relógio híbrido* [13]. Este modelo apresenta um grau de precisão maior para *etl* do que para o *etr*: o *etr* agrega atualizações de outras réplicas (*remotas*), de modo a manter um tamanho de metadados reduzido. Esta

assimetria deve-se a assumirmos que os clientes fazem muito mais operações na réplica a que estão ligados (*operações locais*) e também melhora a sua mobilidade. Como mostraremos, os resultados experimentais comprovam a existência de um bom equilíbrio entre o tamanho dos metadados e a sua precisão.

2.3 Interação com os Clientes

Os clientes interagem com qualquer cúmulo ou centro de dados do sistema. Antes de qualquer leitura ou atualização numa réplica diferente, o cliente precisa de se *ligar* à mesma. O objetivo é sincronizar os estados de ambos. Como resultado do processo de ligação a um cúmulo, o cliente obtém uma etiqueta multi-parte que deve usar sempre que tentar ler ou atualizar dados. Designamos por *migração* o processo de um cliente se desligar de uma réplica para se ligar a outra (por exemplo, para aceder a dados que não se encontram na réplica à qual se encontra ligado). O suporte para a migração de clientes é um requisito que deriva da necessidade de replicação parcial. Estes protocolos serão descritos a adiante. De seguida, descrevemos as operações básicas que os clientes ligados podem efetuar.

Atualização: O cliente atualiza os dados diretamente num cúmulo. Assincronamente, este propaga a atualização na sua região para o centro de dados e outros cúmulos que repliquem esses dados. A réplica cria os novos metadados da atualização, preservando a *etr* do cliente e renovando a *etl*. O cliente guarda-os como dependências. Finalmente, o *exportador*, cuja função será apresentada posteriormente, envia os metadados para o corretor regional.

Leitura: Se os dados estiverem presentes no cúmulo a que o cliente se encontra ligado, a leitura simplesmente retorna a versão mais recente dos dados. Caso contrário, o cliente requer uma migração que, por omissão, é feita para o centro de dados da região. Eventualmente, o cliente recebe uma versão do objeto que é coerente com a sua história causal, que pode ter sido criada dentro ou fora da réplica a que está ligado. Se a versão tiver sido criada localmente, o cliente mantém: $\{etl_{Cl}, etr_{Cl}\} = \{max(etl_{Cl}, etl_{Dados}), max(etr_{Cl}, etr_{Dados})\}$. Se a versão não tiver origem local, o cliente só poderá alterar o seu *etr*.

2.4 O Corretor Regional

O *corretor regional* é central na nossa arquitetura. Ele integra os metadados produzidos nos cúmulos e no centro de dados de uma mesma região, formando uma cadeia única, coerente com a causalidade, que é propagada para todas as réplicas. O corretor não distingue o centro de dados dos cúmulos.

O corretor interage a *jusante* com o centro de dados e cúmulos da sua região e a *montante* com os corretores de outras regiões geográficas. Na gestão de metadados, o corretor é quem representa a sua região, escondendo a existência de cúmulos. Por isso, a adição de novos cúmulos não afeta outras regiões.

Os canais de metadados para outras componentes foram modelados como portos com duas ligações *First In, First Out* (FIFO): uma para *entrada* e outra para *saída*. As mensagens de metadados que entram são colocadas numa fila até que o corretor as leia. Para cada porto, a comunicação externa é feita através da

fila da ligação de saída. Caso uma fila de saída tenha múltiplas mensagens, elas podem ser aglomeradas num único pacote, aumentando a eficiência.

O tipo de transformações que o corretor efetua sobre os metadados depende do porto que recebe as mensagens. Se elas chegaram a jusante, o *etr* da mensagem é substituído por um novo, criado pelo corretor. O novo *etr* posiciona a atualização em relação a todas as outras atualizações recebidas no corretor. Posteriormente, o par *etl* (preservado) e *etr* é reencaminhado para todas as réplicas locais. Para os portos a montante, o corretor envia somente o novo *etr*. Os metadados gerados noutras regiões são inseridos na cadeia regional de metadados. O *etr* é substituído pela posição que a atualização ocupa nesta cadeia. Finalmente, os novos metadados são reenviados para as réplicas locais.

2.5 Operação das Réplicas

Cada réplica mantém dois serviços de suporte (*exportador* e o *importador*), que permitem interagir com o corretor regional e com as restantes réplicas.

Exportador: O *exportador* coleciona os metadados gerados localmente, ordena os mesmos respeitando a causalidade e envia-os para o corretor regional. As técnicas utilizadas seguem o mesmo princípio das apresentadas em [11].

Importador: O *importador* recebe atualizações externas e aplica-as respeitando a coerência causal. Com este objetivo, os dados são visíveis pela ordem de chegada dos metadados, que já respeita esta ordenação. Portanto, o seu funcionamento consiste em ler os metadados que lhe foram enviados, esperar pelos dados associados e aplicar as alterações localmente. Adicionalmente, o importador guarda os metadados mais recentes criados em cada réplica remota, juntamente com o máximo *etr* alguma vez visto, de forma a acelerar a ligação/migração.

2.6 Migração de Clientes

Um cliente *liga-se* a uma réplica antes de poder efetuar leituras e atualizações. O objetivo é garantir a *sincronização* entre ambos.

Ligação: Considerando que o cliente esteve pela última vez ligado à réplica *origem* e está a tentar transferir-se para a réplica *destino* (*origem*$\neq$*destino*), o cliente apresenta a sua etiqueta ao importador de *destino*. O importador aguarda até que a seguinte condição de *migração segura* seja verificada: *tanto o* etl *como o* etr *existentes no destino apresentam valores maiores ou iguais do que os da etiqueta do cliente*. De seguida, o importador cria uma nova etiqueta para o cliente: $\{etl_{Cl}, etr_{Cl}\} = \{0, max(etr_{Cl}, etr_{Destino}[Origem])\}$. Como o cliente se acabou de ligar ao *destino*, ele não apresenta qualquer dependência local (visível pelo valor 0). No entanto, o *etl* na *origem* tem de ser incluído na história causal do cliente através da fusão com o *etr*. Devido ao *destino* ter, no mínimo, as versões que o cliente depende, existe a garantia que o *etr* da dependência local do cliente está presente no *destino*. O cliente deve manter o máximo entre os valores de *etr*, de forma a garantir a correção da sua história causal.

Migração: Por razões de desempenho, os clientes efetuam uma operação de migração na réplica de *origem*, antes de se ligarem ao *destino*. Esta operação

consiste na geração de uma atualização *falsa*, gerando metadados na *origem* que serão enviados para o *destino*. Este passo garante que as estampilhas no *destino* são atualizadas de forma a aceitar a ligação do cliente, mesmo que o *destino* receba poucas atualizações. Note-se que a atualização dos objetos falsos é feita em paralelo com a ligação à réplica de *destino*, para não atrasar o cliente desnecessariamente. Na sua essência, a latência de migração é limitada superiormente pela latência de propagação dos metadados associados à atualização falsa. Se os clientes não contactassem a *origem*, não haveria qualquer garantia quanto à terminação do processo (a não ser que o *destino* fosse o centro de dados). Caso a *origem* falhe, se todas as atualizações locais forem exportadas, o cliente ligar-se-ia diretamente ao centro de dados. Se houver atualizações em falta, ter-se-á de decidir entre não fazer progresso ou uma violação momentânea da causalidade.

3 Avaliação

A nossa avaliação foca-se em aferir o impacto das diferentes estratégias nos seguintes aspetos: i) *latência de visibilidade das atualizações*; ii) *latência de migração*; iii) *débito máximo do sistema*.

De modo a encontrar o melhor compromisso entre tamanho dos metadados e impacto de falsas dependências, decidimos comparar a nossa solução a dois sistemas relevantes que atingem os melhores resultados em cada um destes aspetos. Nomeadamente, comparamos a nossa arquitetura com dois algoritmos adaptados de COPS [15] e Saturn [7]. O COPS é um sistema que mantém dependências explícitas. Apesar da compressão após as atualizações, os clientes podem guardar metadados cuja dimensão é linear com o número de itens do sistema. Portanto, o COPS mantém uma grande quantidade de metadados, mas consegue rastrear a causalidade com precisão. O Saturn utiliza metadados de tamanho constante e usa um serviço de propagação dos metadados que recorre a uma topologia em árvore para minimizar o impacto da perda de precisão. No entanto, o Saturn não foi desenhado para otimizar o processo de migração, uma vez que estas são infrequentes num sistema de computação na nuvem puro. A versão COPS aqui avaliada é um híbrido entre o COPS original e o Saturn. Os clientes guardam as dependências como no primeiro, mas os metadados são propagados como no segundo. Esta alteração permite-nos descobrir qual o tempo mínimo para a migração de clientes em sistemas com coerência causal. Todas as soluções apresentam um procedimento de serialização intra-réplica, como descrito em [11].

A solução aqui proposta usa bastante menos metadados do que COPS, mas mais do que Saturn (Saturn usa o tuplo $\langle tipo, origem, estampilha, alvo\rangle$ e a nossa solução precisa de mais uma *estampilha* e *origem*, para o *etr*). Também não recorre a topologias complexas para a propagação dos metadados: as réplicas organizam-se numa estrela com o corretor no centro. Por outro lado, usamos algoritmos mais sofisticados para gerir os metadados. Por isso, os seguintes resultados devem ser observados experimentalmente: i) o COPS deve exibir a menor latência de visibilidade de atualizações, mas os outros dois sistemas não devem estar muito atrás; ii) o COPS deve permitir migrar mais rapidamente, devendo

a nossa arquitetura aproximar-se deste sistema e superar o Saturn nesta faceta; iii) o débito de operações do sistema aqui proposto deve ser aproximado ao de Saturn, e ambos mais elevados do que o de COPS.

Concretização: Para realizarmos uma avaliação da nossa arquitetura, desenvolvemos um protótipo que concretiza os mecanismos e protocolos aqui descritos. O ponto de partida foi o protótipo do Saturn, ao qual foram introduzidas alterações para suportar não só o nosso sistema (Cúmulos com replicação total e parcial), mas também o algoritmo do COPS com replicação total e um sistema que simula uma migração através da leitura de um objeto no *destino* (coerência eventual). Cúmulos com replicação total simula um sistema para a nuvem, mas a versão parcial é a pensada para a periferia. A linguagem de programação utilizada foi Erlang/OTP. Relógios físicos geram os *etl*s e lógicos criam os *etr*s.

Configuração: Nós utilizamos uma topologia em estrela com corretor e o centro de dados na localização central, rodeados por sete cúmulos. O centro de dados replica todos os itens da região e cúmulos têm replicação parcial. Cada cúmulo pode armazenar grupos independentes de itens. Os clientes estão nas mesmas localizações das suas réplicas locais originais, simulando uma proximidade geográfica. Todas as experiências deste artigo foram feitas através da infraestrutura de Grid'5000 [2]. O centro de dados foi colocado em Lyon, por ser a localização central. As restantes localizações têm um cúmulo próprio. A Figura 2 (esquerda) apresenta as latências médias entre os nós do sistema. Cada máquina tem especificações próprias: 2 processadores físicos com 4 a 8 núcleos; memória entre 4 e 64 GB; sistema operativo Ubuntu 14.04. Optámos por usar uma máquina com mais recursos como centro de dados e máquinas com menos recursos nos locais que representam os cúmulos, de forma a aproximar a diversidade de máquinas que existe em cenários de computação na periferia. A bancada experimental não permite emular com facilidade a diversidade de condições de rede que caracteriza a computação na periferia, uma vez que todos os clientes se executam nos nós da Grid'5000. Os clientes testam os sistemas através de uma versão adaptada de Basho Bench [1]. Cada avaliação corre durante 2 minutos e tanto os primeiros como os últimos 10 segundos são descartados. A latência de visibilidade de atualizações é calculada no destino, pela diferença entre o momento atual e o *etl* da atualização. Para sincronizar os relógios, todas as máquinas seguem o protocolo NTP [3] antes de começar os testes. Todos os outros resultados são obtidos no lado do cliente.

Cargas: As cargas são sintéticas e consideram três comportamentos de clientes: i) só operações locais, com 90% de leituras e 10% de atualizações; ii) uma mistura de operações locais e migrações, com uma distribuição de 70% de leituras, 10% de atualizações e 20% de migrações para qualquer réplica; iii) maioritariamente migrações, com 90% de migrações e 10% de atualizações.

3.1 Operações Locais

Para esta experiência, focamos-nos em clientes que só fazem operações locais. Os resultados da Figura 2 (direita) mostram que COPS apresenta o pior desempenho, por preceder as atualizações por uma verificação das suas dependências.

	Li	Lu	Ly	Nc	Nt	R	S
G	7,7	9,7	1,5	8,7	8,0	8,6	4,7
Li	-	5,7	6,6	4,8	13,0	13,6	9,7
Lu	-	-	8,5	1,2	15,0	15,6	11,7
Ly	-	-	-	7,5	6,9	7,4	3,5
Nc	-	-	-	-	14,0	14,6	10,7
Nt	-	-	-	-	-	0,8	10,0
R	-	-	-	-	-	-	10,6

		Média	95%	99%
Saturn	**L**	0,7	1,1	2,4
	A	0,8	1,2	2,4
COPS	**L**	0,9	2,0	4,2
	A	1,9	4,2	6,8
Cúmulos Total	**L**	0,7	1,2	3,0
	A	0,8	1,4	3,2
Cúmulos Parcial	**L**	0,7	1,0	2,3
	A	0,8	1,2	2,6

Figura 2. Latências médias em *ms* entre as localizações Grid'5000 [2]: Grenoble (G), Lille (Li), Luxembourg (Lu), Lyon (Ly), Nancy (Nc), Nantes (Nt), Rennes (R) e Sophia (S). Comparação da latência de operações locais: leituras (L) e atualizações (A).

Isto aumenta a carga na réplica e atrasa todas as operações. Os outros três sistemas têm valores médios semelhantes. No pior caso, a nossa arquitetura com replicação total é um pouco mais lenta. Isto é um artefacto da experiência, que se deve a todos os itens existirem em todas as réplicas, permitindo uma maior liberdade de movimentos aos clientes. Algumas vezes, a quantidade de clientes ligados a uma mesma réplica vai além do que é suportado, aumentando as filas de pedidos. Em simultâneo, há mais atualizações remotas recebidas nos cúmulos.

3.2 Visibilidade de Atualizações Remotas

Para esta experiência, verificamos se as alterações influenciam a latência de visibilidade das atualizações. Na Figura 3 (esquerda), todas as soluções apresentam resultados semelhantes. Sistemas com replicação total são um pouco mais lentos, devido à concentração de clientes em determinadas réplicas. Um resultado interessante é o da atualização mais rápida ocorrer após cinco milissegundos, apesar de existir uma latência média mínima de $0,8$ milissegundos. As causas são o *processo de estabilização local* do Eunomia [11] e a aglomeração de atualizações a exportar. Em média, as atualizações são enviadas após quatro milissegundos.

3.3 Migração dos Clientes

Como referimos, em cenários de computação na periferia é fundamental que a migração de clientes entre réplicas possa ser feita de forma muito eficiente. A Figura 3 (direita) mostra o desempenho dos diferentes sistemas no suporte à migração dos clientes. Tal como esperado, o sistema com coerência eventual é o mais rápido a concluir a migração. Desta vez, os clientes Saturn são os mais lentos, por causa da constante criação de metadados para migração. COPS captura o limiar mínimo para os sistemas com coerência causal. A nossa solução com replicação total apresenta resultados idênticos, porque todas as réplicas receberão todas as atualizações, não aguardando pela atualização do objeto falso. A replicação parcial aumenta a latência, mas obtemos resultados competitivos para réplicas que se encontrem geograficamente próximas (a mediana é somente

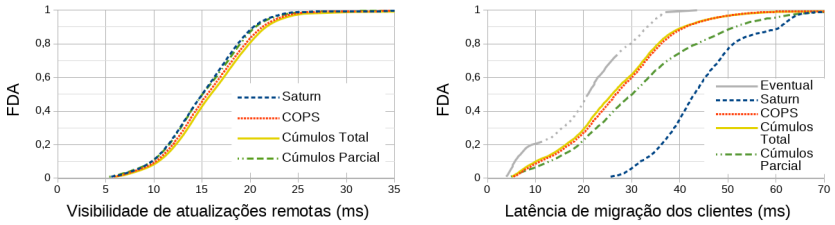

Figura 3. Comparação das latências de visibilidade e de migração.

cinco milissegundos mais lenta). Em comparação com o Saturn, a nossa solução completa cerca de 80% das migrações enquanto o primeiro ainda só vai a metade, pois o cliente migra com metadados que foram previamente replicados.

3.4 Débito de Operações

Finalmente, queremos manter um bom débito de operações (quantidade de operações por unidade de tempo). A nossa solução com replicação total apresenta o maior débito de operações, sendo seguida pela mesma com replicação parcial (diferença de $0,3\%$). A concentração de clientes e a diferença entre as migrações não ser significativa para réplicas próximas reduzem o intervalo. Saturn é o terceiro (cerca de 5% menos do que a nossa solução com replicação parcial), por causa do seu demorado processo de migração. No entanto, não sobrecarrega os servidores, possibilitando rápidas operações locais. O COPS apresenta muito boas latências, mas o seu débito de operações é o mais baixo (20% menos do que a nossa arquitetura com replicação parcial). A verificação explícita de dependências desvia recursos de computações úteis.

4 Trabalho Relacionado

Nos últimos anos, foram desenvolvidos muitos trabalhos relacionados com o armazenamento na nuvem oferecendo coerência causal. Fazemos um apanhado dos trabalhos mais relevantes, referindo primeiro a quantidade de metadados que estes mantêm e posteriormente a sua capacidade de suportar replicação parcial.

O sistema COPS [15] rastreia as dependências de forma explícita. A quantidade de metadados é proporcional ao número total de itens. Na pior das situações, os metadados podem conter uma entrada por cada item, reduzindo o desempenho do sistema. No entanto, a precisão de rastreamento das dependências é alta, conduzindo a poucas falsas dependências e baixa latência de visibilidade de atualizações. Outras soluções tentam manter menos metadados. O Orbe [9] codifica os metadados como uma *matriz de dependências* (uma entrada por partição por réplica); os sistemas PRACTI [6], SwiftCloud [24], Legion [14] e Eunomia [11] utilizam vetores de versão com uma entrada por réplica. GentleRain [10] tem metadados de tamanho constante. O Saturn [7] foi o primeiro

sistema a conseguir reduzir a latência da visibilidade de atualizações com poucos metadados, mantendo um débito de operações semelhante a GentleRain. A causa é a inovadora propagação de metadados baseada numa estrutura em árvore. O nosso sistema recorre a uma estrela, pois as réplicas estão próximas e assim evitamos o peso de reconfigurar a árvore na presença de faltas.

O suporte para replicação parcial tem duas variantes: *genuína* e *não-genuína*. Replicação parcial genuína evita que as réplicas recebam metadados sobre itens que não guardam, ao contrário de replicação parcial não-genuína. O PRACTI [6] é uma exemplo de replicação parcial não-genuína. As réplicas recebem *mensagens de invalidação* sobre itens singulares ou grupos de itens. O Saturn [7] oferece replicação parcial genuína. Ao propagar os metadados pela árvore, os nós sabem para onde os reencaminhar. Este conhecimento evita que os metadados atinjam réplicas não interessadas. A nossa solução é inspirada no Saturn. Duas soluções que consideram certos dispositivos de periferia são o SwiftCloud [24] e o Legion [14]. O primeiro fixa a sessão a uma réplica e o segundo não consegue garantir consistência causal entre *contentores* (agrupamento de objetos) diferentes.

5 Conclusões e Trabalho Futuro

Neste artigo apresentamos uma arquitetura de armazenamento de dados que visa providenciar coerência causal na periferia da rede. Desenvolvemos um protótipo desta arquitetura que usamos para aferir o seu desempenho. Uma grande vantagem da nossa arquitetura é permitir que os clientes migrem entre cúmulos de forma eficiente. Isto é importante para a computação na periferia da rede, visto que os itens estão espalhados entre diferentes cúmulos e os cliente móveis podem querer ligar-se a diferentes cúmulos ao longo de uma sessão. A solução aqui proposta suporta uma taxa de migrações que é uma ordem de magnitude superior à do Saturn, um dos sistemas de referência para replicação parcial. No futuro, iremos avaliar os protocolos de reconfiguração do sistema. Para demonstrar a utilidade do trabalho apresentado, gostaríamos também de o integrar numa aplicação real.

Agradecimentos Agradecemos aos revisores e a Diogo Silva pelos comentários recebidos durante a preparação deste artigo. Este trabalho foi parcialmente suportado pela Fundação para a Ciência e Tecnologia (FCT) através dos projetos com referências PTDC/EEI-COM/29271/2017 (Cosmos) e UID/ CEC/ 50021/ 2013.

Referências

1. Basho Bench, http://github.com/basho/basho_bench, Acedido: 2018-04-20
2. Grid'5000, https://www.grid5000.fr, Acedido: 2018-04-20
3. NTP: The Network Time Protocol, http://www.ntp.org/, Acedido: 2018-04-20
4. Ahmeda, E., Rehmani, M.: Mobile Edge Computing: Opportunities, solutions, and challenges. Pervasive Computing (70), 59–63 (2017)

5. Ai, Y., Peng, M., Zhang, K.: Edge computing technologies for Internet of Things: a primer. Digital Communications and Networks **4**(2), 77 – 86 (2018)
6. Belaramani, N., Dahlin, M., Gao, L., Nayate, A., Venkataramani, A., Yalagandula, P., Zheng, J.: PRACTI Replication. In: NSDI'06. San Jose, CA, USA (2006)
7. Bravo, M., Rodrigues, L., Van Roy, P.: Saturn: A Distributed Metadata Service for Causal Consistency. In: EuroSys '17. Belgrade, Serbia (2017)
8. Bravo, M., Diegues, N., Zeng, J., Romano, P., Rodrigues, L.: On the use of Clocks to Enforce Consistency in the Cloud. IEEE Data Engineering Bulleting **38**(1) (2015)
9. Du, J., Elnikety, S., Roy, A., Zwaenepoel, W.: Orbe: Scalable Causal Consistency Using Dependency Matrices and Physical Clocks. In: SOCC '13. Santa Clara, CA (2013)
10. Du, J., Iorgulescu, C., Roy, A., Zwaenepoel, W.: GentleRain: Cheap and Scalable Causal Consistency with Physical Clocks. In: SOCC '14. Seattle, WA, USA (2014)
11. Gunawardhana, C., Bravo, M., Rodrigues, L.: Unobtrusive Deferred Update Stabilization for Efficient Geo-replication. In: USENIX ATC '17. Santa Clara, CA (2017)
12. Hao, P., Bai, Y., Zhang, X., Zhang, Y.: EdgeCourier: An Edge-hosted Personal Service for Low-bandwidth Document Synchronization in Mobile Cloud Storage Services. In: SEC 2017. San Jose, CA (2017)
13. Kulkarni, S.S., Demirbas, M., Madappa, D., Avva, B., Leone, M.: Logical Physical Clocks. In: OPODIS '14. Cortina d'Ampezzo, Italy (2014)
14. Linde, A.v.d., Fouto, P., Leitão, J., Preguiça, N., Castiñeira, S., Bieniusa, A.: Legion: Enriching Internet Services with Peer-to-Peer Interactions. In: WWW '17 Companion. Perth, Australia (2017)
15. 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: SOSP '11. Cascais, Portugal (2011)
16. Mehdi, S.A., Littley, C., Crooks, N., Alvisi, L., Bronson, N., Lloyd, W.: I Can'T Believe It's Not Causal! Scalable Causal Consistency with No Slowdown Cascades. In: NSDI'17. Boston, MA (2017)
17. Mortazavi, S., Salehe, M., Gomes, C., Phillips, C., de Lara, E.: Cloudpath: A Multi-tier Cloud Computing Framework. In: SEC 2017. San Jose, CA (2017)
18. Patel, M., Hu, Y., Heédeé, P., Joubert, J., Thornton, C., Naughton, B., Ramos, J., Chan, C., Young, V., Tan, S., Lynch, D., Sprecher, N., Musiol, T., Manzanares, C., Rauschenbach, U., Abeta, S., Chen, L., Shimizu, K., Neal, A., Cosimini, P., Pollard, A., Klas, G.: Mobile-Edge Computing. Tech. rep., ETSI (Sep 2014)
19. Ricart, G.: A City Edge Cloud with its Economic and Technical Considerations. In: SmartEdge 2017. Kona (HI), USA (2017)
20. Satyanarayanan, M.: A Brief History of Cloud Offload. ACM/SIGMOBILE GetMobile Magazine **18**(4), 19–23 (2014)
21. Satyanarayanan, M., Bahl, P., Cáceres, R., Davies, N.: The Case for VM-Based Cloudlets in Mobile Computing. Pervasive Computing **8**(4), 2–11 (2009)
22. Streiffer, C., Srivastava, A., Orlikowski, V., Velasco, Y., Martin, V., Raval, N., Machanavajjhala, A., Cox, L.: ePrivateEye: To the Edge and Beyond! In: SEC 2017. San Jose, CA (2017)
23. Tomsic, A.Z., Crain, T., Shapiro, M.: Scaling Geo-replicated Databases to the MEC Environment. In: SRDSW '15. Montreal, Quebec, Canada (2015)
24. Zawirski, M., Preguiça, N., Duarte, S., Bieniusa, A., Balegas, V., Shapiro, M.: Write Fast, Read in the Past: Causal Consistency for Client-Side Applications. In: Middleware '15. Vancouver, BC, Canada (2015)