Avaliação das Garantias de Consistência em Serviços Geo-Replicados

João Costa, João Leitão, Daniel Porto, Nuno Preguiça, and Rodrigo Rodrigues

CITI — Departamento de Informática,
Universidade Nova de Lisboa, Portugal
jp.costa@campus.fct.unl.pt, jc.leitao@fct.unl.pt, d.porto@campus.fct.unl.pt,
nuno.preguica@di.fct.unl.pt, rodrigo.rodrigues@fct.unl.pt

Resumo Com o aumento da popularidade de serviços distribuídos que recorrem à geo-replicação, a comunidade científica tem efectuado um esforço activo para desenvolver modelos de consistência e esquemas de replicação, que permitam a estas aplicações encontrar um equilíbrio adequado entre desempenho e a exposição da camada de replicação para os utilizadores destas aplicações. No entanto, é pouco claro quais os modelos de consistência que são oferecidos por aplicações reais e extremamente populares, como por exemplo o Facebook ou o Twitter. Neste artigo, propomos uma metodologia e descrevemos uma arquitectura que pretende validar um conjunto de propriedades relevantes relativas ao modelo de consistência oferecido por aplicações reais de grande escala. Em particular a nossa abordagem permite verificar violações de propriedades de sessão bem conhecidas, assim como inferir a Janela de Divergência máxima observada pelos clientes. O sistema alvo é visto como uma caixa negra, não sendo necessário qualquer conhecimento base sobre a operação interna do mesmo, e baseia-se num conjunto de dois testes que propomos no artigo.

Keywords: Geo-replicação, Modelos de consistência, Caracterização de sistemas

1 Introdução

O número e popularidade de serviços distribuídos na Internet sofreram um crescimento significativo em anos recentes. Entre os exemplos que evidenciam este facto, encontra-se a rede social Facebook, o serviço de *micro-blogging* Twitter, a plataforma de partilha de imagens Flickr, entre outros. De forma a garantir a escalabilidade, disponibilidade, e também para minimizar os tempos de resposta aos utilizadores, muitos destes serviços recorrem a geo-replicação, de forma a que os dados mantidos por estes serviços - e manipulados pelos utilizadores - se encontram replicados em várias localizações (tipicamente centros de dados privados) - espalhados pelo planeta.

De forma a tentar ocultar a existência de replicação dos utilizadores, em particular da existência de várias cópias do mesmos itens de dados disponíveis,

as quais podem apresentar diferentes valores devido ao tempo necessário para propagar as operações dos utilizadores entre as várias réplicas, a comunidade de sistemas distribuídos tem efectuado um esforço significativo para desenvolver modelos de consistência de dados e esquemas de replicação que implementem esses mesmos modelos, de forma a minimizar a exposição da replicação, e simultaneamente oferecer um desempenho adequado para estes sistemas.

Neste contexto, seria desejável a existência de mecanismos ou ferramentas disponíveis à comunidade em geral que permitissem validar de uma forma simples as garantias de consistência que são fornecidas efectivamente por um sistema. Tal permitiria não só a validação de implementações de sistemas replicados, como também o teste de sistemas em linha (*online*) existentes. Para endereçar a ausência de tais ferramentas, neste artigo fazemos as seguintes contribuições: i) identificamos um conjunto de propriedades fundamentais de consistência que estão presentes em vários modelos de consistência propostos na literatura; ii) apresentamos uma metodologia que permite verificar se estas propriedades fundamentais são violadas por um sistema em particular; esta metodologia não necessita de qualquer informação sobre o funcionamento interno do sistema alvo - observando o sistema como uma caixa negra - e assenta num conjunto de dois testes que exercitam o sistema através das interfaces oferecidas ao clientes utilizando vários processos espalhados pelo mundo; iii) apresentamos uma ferramenta que serve de pedra angular da metodologia proposta e que implementa os testes anteriormente referidos, e finalmente, iv) apresentamos resultados experimentais *preliminares*, nos quais aplicamos esta ferramenta para validar as propriedades fundamentais do Google+, uma rede social popular.

O resto deste artigo está organizado da seguinte forma. De seguida, a Secção 2 identifica algumas propriedades fundamentais de vários modelos de consistência. A Seccção 3 apresenta a nossa metodologia para a validação de modelos de consistência, descrevendo em detalhe os dois testes fundamentais centrais à nossa abordagem. A Seccção 4 discute alguns detalhes da nossa implementação e aprensenta resultados preliminares em que recorremos à nossa metodologia para avaliar as propriedades de consistência oferecidas pelo Google+. O trabalho relacionado é discutido na Secção 5, e finalmente, concluímos este artigo identificando direções para trabalho futuro na Secção 6.

2 Propriedades de consistência

Existem vários modelos de consistência propostos na literatura de algoritmos distribuídos que definem, intuitivamente, um conjunto de garantias que são oferecidas aos clientes relativamente à visão que estes têm do estado do sistema, em relação as suas operações passadas, e também relativamente a dependência que possam existir entre essas operações.

Neste trabalho, em vez de nos focarmos em aferir um sub-conjunto das múltiplas definições propostas na literatura, decidimos focar num conjunto pequeno de propriedades mais fundamentais, em que cada propriedade pode

estar presente em mais do que uma definição de consistência. No final desta secção providenciamos exemplos concretos que ilustram a utilização destas propriedades fundamentais para a extração de conclusões relativas ao modelo de consistência oferecido por um sistema.

A nossa apresentação divide estas propriedades em duas classes: Garantias de Sessão e Garantias entre Sessões. De seguida oferecemos uma descrição de cada uma desta classes, e discutimos as propriedades fundamentais que consideramos no contexto deste artigo para cada uma das classes.

2.1 Garantias de sessão

No contexto de um sistema replicado, uma sessão é uma abstração que captura um conjunto de operações efectuadas por um único cliente em sequência. As garantias de sessão são por isso garantias oferecidas relativamente ao estado do sistema que um cliente observa considerando apenas as operações efectuadas por esse cliente anteriormente. As garantias de sessão que consideramos foram definidas no passado por Terry et al. [1].

Em particular, as garantias de sessão que consideramos neste artigo são: *Read Your Writes*, *Monotonic Reads*, *Writes Follow Reads* e *Monotonic Writes*. Note-se que estas garantias foram definidas no contexto do desenvolvimento de um sistema replicado com consistência fraca denominado Bayou [4], onde os autores argumentam que a combinação de todas estas garantias permite oferecer aos clientes um visão intuitiva do estado do sistema, que emula um sistema sem qualquer forma de replicação.

De seguida fornecemos as definições destas garantias que consideramos neste artigo, assim como os cenários que indicam que a garantia é violada, e que servirão de base aos mecanismos que descreveremos para testar a presença ou não de cada propriedade.

Read Your Writes: A garantia de *Read Your Writes* estabelece que os resultados de uma operação de escrita têm de ser visíveis por qualquer operação de leitura posterior efectuada na mesma sessão. Informalmente, esta propriedade assegura que um utilizador irá ver sempre os efeitos das suas operações de escrita. Caso uma leitura de um cliente c retornar um valor anterior ao escrito pela última operação de escrita de c, a propriedade é considerada violada.

Monotonic Reads: A garantia de *Monotonic Reads* estabelece que uma leitura de um cliente apenas pode retornar um valor que reflita os efeitos de todas as escritas observadas pela última leitura desse mesmo cliente na mesma sessão. Informalmente, esta propriedade assegura que nenhuma leitura de um cliente pode regressar a um estado do sistema considerado passado relativo a outras operações de leitura na mesma sessão. Se um cliente c observar um estado do sistema s gerado por uma operação de escrita op, e numa operação subsequente de leitura observar um estado s' com $s' < s$, considera-se que esta propriedade é violada.

Writes Follows Reads: A garantia de *Writes Follows Reads* estabelece que qualquer operação de escrita que ocorra após uma operação de leitura de um mesmo cliente, apenas terá efeito após a execução de todas as operações de escrita cujos efeitos constavam do estado retornado nessa operação de leitura. Informalmente, esta propriedade requer que qualquer operação de escrita de um cliente c seja serializada após todas as operações de escrita cujos efeitos foram refletidos na última leitura desse mesmo cliente. Esta propriedade é violada em situações em que os efeitos de uma escrita de um cliente não sejam observados por operações subsequentes desse cliente na ausência de escritas concorrentes.

Monotonic Writes: A garantia de *Monotonic Writes* estabelece que os efeitos de qualquer operação de escrita de um cliente c sejam aplicados sobre um estado do sistema que reflita todos os efeitos de todas as escritas anteriores de c. Informalmente, qualquer operação de escrita de um cliente deve ser serializada após todas as operações de escrita efectuadas anteriormente por esse mesmo cliente na mesma sessão. Esta propriedade é violada no caso em que um cliente c executa um conjunto de operações de escrita que se sobrepõem na totalidade aos efeitos das escritas anteriores, e em que os efeitos da última escrita nunca são observáveis após todas as escritas terem sido propagadas e tornadas visíveis aos restantes clientes, mas sim os efeitos de uma das escritas anteriores.

2.2 Garantias entre Sessões

As garantias entre sessões capturam garantias relativas às dependências de operações entre sessões de clientes diferentes. Neste contexto consideramos duas propriedades fundamentais:

Causalidade: A causalidade estabelece que um cliente c apenas deve observar um estado do sistema que contenha todos os efeitos observados ou criados por operações anteriores desse cliente, e transitivamente que observe todos os efeitos observados ou criados por operações de escrita efectuadas por outros clientes cujos efeitos foram já observados por c. Uma violação desta propriedade ocorre quando um cliente a observa os efeitos de uma operação op_1 e após esta observação efectua uma operação op_2, e um cliente b observa os efeitos da operação op_2 sem no entanto observar os efeitos da operação op_1 da qual op_2 *depende causalmente*.

Janela de Divergência: A Janela de Divergência é definida como o intervalo de tempo entre o final da execução de uma operação de escrita op por um cliente c (*i.e.*, quando o sistema retorna uma resposta a c) e o instante a partir do qual é garantido que qualquer outro cliente c' observa os efeitos de op. Isto reflecte por isso o imediatismo de uma dada operação, reflectindo o tempo máximo que um cliente demorar sem observar os efeitos de qualquer operação de escrita que ocorra no sistema. Sendo esta uma propriedade quantitativa que estudamos, não existe uma violação da mesma – podemos apenas dizer que numa execução em que a escrita de um cliente c_1 demora

t unidades de tempo a reflectir-se nas leituras de um cliente c_2, a Janela de Divergência é igual ou superior a t.

2.3 Discussão

Como mencionado, estas propriedades fundamentais são partes constituintes de definições de consistência mais elaboradas. Como tal, o facto de algumas destas propriedades serem violadas permite imediatamente detectar a violação de alguns modelos de consistência encontrados na literatura. Consideremos a propriedade de causalidade por exemplo, a violação desta propriedade imediatamente implica que o modelo de consistência causal não é respeitado pelo sistema a ser analisado. De forma análoga uma janela de divergência não nula implica que a consistência que o sistema oferece não obedece à linearizibilidade. Uma discussão mais detalhada destas implicações é ortogonal às contribuições principais deste artigo, e será abordada em detalhe em trabalho futuro.

3 Metodologia

3.1 Visão geral

Por forma a verificar a eventual violação das garantias individuais identificadas na secção anterior, recorremos a um conjunto de agentes, executados a partir de várias localizações do mundo, tentando assim garantir que agentes em localizações diferentes sejam servidos por centros de dados distintos. Tal pode potenciar que diferentes grupos de agentes acedam a réplicas distintas do estado do serviço. Recorremos a fornecedores de computação em nuvem por forma a localizar estes agentes em regiões distintas do globo. Estes agentes executam pedidos ao serviço cujas garantias de consistência estão a ser testadas, recorrendo a bibliotecas cliente tipicamente oferecidas por esses serviços.

Apesar de cada serviço possuir interfaces específicas suportando operações específicas à natureza do serviço, no contexto de cada agente apenas distinguimos entre dois tipos de operação. Operações de *escrita* que modificam o estado do serviço de uma forma que deve ser visível aos restantes agentes, e operações de leitura, que apenas verificam o estado do serviço sem o modificar. Considerando esta separação, os agentes dividem-se em dois tipos: *Leitores* que apenas efectuam operações de leitura com uma frequência f sobre um subconjunto do estado do serviço e (f e e são parâmetros configuráveis de cada agente individual); e *Escritores* que efectuam operações de leitura similares às dos Leitores, mas adicionalmente efectuam também um conjunto de operações de escrita sobre um subconjunto do estado do serviço e, quando uma condição c é verdade. Esta condição pode ser definida em termos de um instante pré-defenido, ou o facto de que os efeitos de outra escrita tornaram-se visíveis para o agente (*i.e.*, foram retornados numa leitura anterior executada por este agente). Note-se que apesar de um Escritor ter a capacidade de, no contexto de um único teste, executar várias operações de escrita, estas não ocorrem concorrentemente.

A metodologia que propomos é fundamentalmente baseada em dois testes executados colaborativamente por um conjunto de agentes. O primeiro tem como objectivo validar as garantias de *Write Follows Reads*, *Monotonic Writes* e *causalidade*. O segundo teste visa verificar as restantes garantias nomeadamente, *Monotonic Reads*, *Read Your Writes* e finalmente a propriedade de Janela de Divergência. Note-se que agregamos a validação de várias garantias num único teste por forma a simplificar o processo de análise de consistência de um serviço alvo e por forma a reduzir o tempo e o custo monetário (em termos de instâncias em infra-estruturas de computação em nuvem) associados ao teste. Esta metodologia oferece também uma forma natural de extensibilidade à validação de outras garantias ou propriedades, bastando para isso incluir novos testes desenhados para esse fim concreto.

Nestes testes, o número (e localização) dos agentes de escrita é fixo, ou simplesmente uma função linear do número de localizações distintas onde os agentes executam.O número de Leitores é um parâmetro configurável, sendo que um maior número de agentes deste tipo, para além de permitir obter medições mais rigorosas, aumenta a probabilidade de ser detectada uma violação de garantia. No entanto, um maior número de Leitores aumenta a complexidade tanto na execução dos testes como na posterior análise de resultados.

Finalmente, muitos serviços distribuídos sobre a Internet contêm a noção de utilizador. Em tais casos, todas as operações sobre o serviço são executadas em nome de um utilizador, e a identidade do utilizador restringe a visibilidade que um agente que actua em nome desse utilizador pode ter do estado partilhado do serviço. Para lidar com este cenário comum, os nossos agentes são configurados por forma a agirem no nome de um utilizador específico. Para simplificar o desenho dos testes, assumimos que os utilizadores em nome dos quais os agentes actuam criaram (fora de banda) permissões específicas no serviço de forma a permitir a visibilidade das suas operações a todos os utilizadores utilizados pelos restantes agentes envolvidos nos testes.

De seguida descrevemos em maior detalhe o desenho de cada um dos testes utilizados no nosso sistema.

3.2 Primeiro teste

O primeiro teste tem como objectivo principal validar a garantia de causalidade. A intuição associada ao desenho deste teste baseia-se em simular uma conversa entre um conjunto de agentes. Assim, considerando que existem N locais no planeta onde os agentes executam (*i.e.*, N centros de dados diferentes), existe um conjunto de escritores E com N escritores, em que $E = \{e_1, .., e_n\}$, em que cada escritor é posicionado numa localização distinta. Para além disso existem Leitores em todos os centros de dados. O número de Leitores é configurável, sendo que no mínimo deve existir um Leitor em cada localização.

O teste é conduzido da seguinte forma. A partir de um instante t todos os Leitores começam a ler com uma frequência f (note-se que os Escritores também agem como Leitores), e o Escritor e_1 realiza uma operação de escrita w_1, cujo efeito pode ser observado por todos os restantes agentes. De seguida

o Escritor efectua operações de Leitura até que a sua própria escrita se torne visível. Quando esta condição for verdadeira, o Escritor e_1 efectua uma nova operação de escrita w_2. Quando o Escritor e_2 observa os efeitos da escrita w_2, este repete os mesmos passos do escritor e_1 efectuando, respectivamente, as operações de escrita w_3 e w_4. Este processo é repetido em sequência por todos os escritores até que o Escritor e_n observa os efeitos da escrita w_{2n-2}, efectuando de seguida as operações de escrita w_{2n-1} e w_{2n}. O objectivo destas escritas em cadeia é criar uma dependência causal dos efeitos de qualquer escrita w_i relativamente a todas as escritas w_j em que $j < i$.

Para que uma operação de escrita ocorra, apenas é necessário que os efeitos da operação de escrita anterior se tornem visíveis ao escritor responsável por realizar essa escrita. No entanto, de forma a verificar se a relação causal entre operações é respeitada, cada agente (Leitores e Escritores) tem o cuidado de verificar, considerando a escrita mais recente observável, se os efeitos de todas as operações de escrita da qual essa escrita depende são também eles observáveis. No entanto, por forma a optimizar o desempenho dos agentes, os agentes guardam apenas os valores retornados em cada leitura num ficheiro local, cuja análise posterior valida se a garantia de causalidade foi violada.

Uma vez que cada Escritor efectua 2 operações de escrita intercaladas com uma operação de leitura que observa os efeitos da primeira escrita, este teste oferece a capacidade a cada Leitor de verificar se a garantia de *Monotonic Writes* é violada. Tal acontece se as duas escritas efectuadas por um único escritor forem observadas por qualquer Leitor por uma ordem inversa à sua ordem de execução. Por outras palavras, considerando um Escritor i, se os efeitos da operação de escrita w_{2i} se tornarem visíveis para um Leitor l antes dos efeitos da operação w_{2i-1}, esta garantia é violada. Para validar a garantia de *Writes Follows Reads*, cada Leitor verifica se este observa os efeitos da primeira operação de escrita de qualquer Escritor w_{2i-1} antes da operação de escrita imediatamente anterior (i.e., w_{2i-2}) na sequência de escritas, caso que indica uma violação desta garantia.

Note-se que a violação das garantias de *Monotonic Write* e *Writes Follows Reads* implica uma violação das garantias de Causalidade.

3.3 Segundo teste

O segundo teste tem como objectivo permitir a medição da Janela de Divergência e verificar as garantias *Monotonic Reads* e *Read Your Writes*.

De forma similar ao teste introduzido anteriormente, e considerando que existem N locais no planeta onde os agentes executam (*i.e.*, N centros de dados diferentes), existe um conjunto de escritores E com N escritores, em que $E = \{e_1, .., e_n\}$, e cada cada escritor é posicionado numa localização distinta. Para além disso existem Leitores em todos os centros de dados. O número de Leitores é configurável, sendo que no mínimo deve existir um Leitor em cada localização.

Neste teste, cada Escritor e_i recebe na sua configuração uma estampilha temporal (*timestamp*) t_i. Em t_i, o Escritor e_i efectua duas operações de escrita

em sequência w_{2i-1} e w_{2i}. Após a conclusão destas escritas esse escritor efectua uma leitura. Os timestamps são atribuídos aos escritores em sequência de forma a que $t_i < t_{i+1}$. Todos os Leitores efectuam leituras com uma frequência f, e registam o seu timestamp local no final de cada leitura individual (este teste assume que os relógios são sincronizados via NTP entre as máquinas que alojam cada agente). Uma vez que o timestamp da escrita é conhecido, isto permite a todos os Leitores inferirem a sua Janela de Divergência para cada escrita efectuada durante o teste. Com base nesta informação, e como iremos mostrar na próxima secção, podemos calcular a Janela de Divergência do serviço considerando os agentes em todas as localizações, os agentes colocados com o agente que efectua cada escrita, e os agentes em cada uma da restantes localizações.

Adicionalmente, este teste permite validar as garantias de *Monotonic Reads* e *Read Your Writes* da seguinte forma. Para verificar a propriedade de *Monotonic Reads*, todos os Leitores verificam entre cada uma das suas leituras que os efeitos das escritas observados na leitura prévia são ainda verificados. Caso contrário considera-se que a garantia é violada. Para validar a garantia de *Read Your Writes*, todos os Escritores devem observar na leitura efectuada após as suas escritas os efeitos de ambas as suas escritas. Caso isto não suceda, considera-se que esta garantia é violada.

4 Avaliação experimental

4.1 Implementação

Para a avaliação da metodologia e da ferramenta aqui proposta, estas foram aplicadas ao serviço Google+. Para tal foi desenvolvido um cliente capaz de cumprir as funções tanto de agente de leitura como de escrita. Quando configurado como agente de leitura, a função do agente é executar operações de leitura sucessivas, sendo estas executadas com uma periodicidade de aproximadamente de $150ms$.

Valores mais reduzidos para a frequência poderiam oferecer maior precisão, no entanto o valor escolhido foi motivado pelo *overhead* associado a estas operações.

Ao obter os resultados das leituras, estes são guardados em disco com indicação do conteúdo lido, bem como da hora a que operação foi executada e a hora a que ela retornou. Quando configurado como agente de escrita, o cliente continuaria a efectuar operações de leitura, mas efectuaria operações de leitura quando apropriado (i.e., na hora assinalada num ficheiro de *input*).

Por as operações de escrita e de leitura serem efectuadas em *threads* diferentes, recorreu-se a um mecanismo de trincos (*locks*) para que as operações de escrita, geralmente compostas por duas operações consecutivas, ocorram de forma atómica. Este mecanismo permite ainda que estas operações só ocorram entre leituras, só podendo ser executadas após a última leitura efectuada ter retornado. A próxima operação de leitura apenas é desbloqueada quando a corrente operação de escrita retornar.

4.2 Configuração

Esta metodologia foi testada com vários agentes hospedados em várias instâncias do serviço Amazon EC2. Em cada teste foram utilizadas três regiões de diferentes continentes (localizadas nos Estados Unidos, Irlanda e Japão), sendo que em cada região havia três agentes, cada um a ser executado numa máquina virtual diferente. Destes três agentes apenas um era agente de escrita, sendo os restantes agentes de leitura. Estas instâncias eram do tipo *Micro*, com 613MB de memória e uma arquitectura AMD64.

Aquando da decisão sobre em que sistema a metodologia iria ser testada, uma dos factores de escolha seria a simplicidade de efectuar as operações e a simplicidade do sistema em si. Inicialmente, a escolha recaiu sobre o Twitter: este serviço oferece uma API REST em que efectuar operações de escrita e de leitura é relativamente simples. No entanto ao efectuar um *traceroute*, a partir de agentes colocados em diversos data-centers geograficamente distantes, verificou-se que os pedidos iriam sempre ser recebidos pelo mesmo servidor, pondo assim em causa a importância da localização dos agentes.

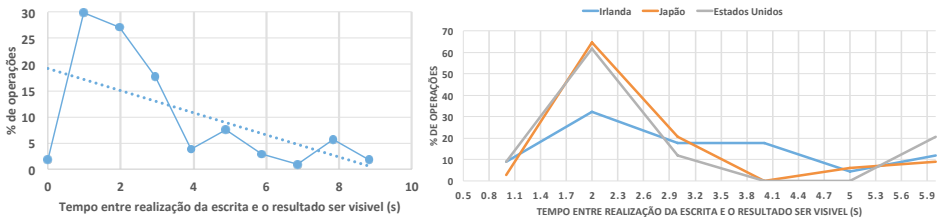
Optou-se então por utilizar outro serviço, sendo este o Google+. As operações são efectuadas com recurso a *Moments* que, no contexto do Google+, representam uma dada operação que uma aplicação faz sobre um determinado perfil. Apesar de estas operações não serem visíveis no perfil do utilizador, este serviço permite listar todos os *Moments* efectuados num dado utilizador. É ainda oferecida uma função de escrita de *Moments*, e ainda uma operação, não utilizada nos testes, para eliminar *Moments*. A funcionalidade *Moments* é similar à funcionalidade de um utilizador adicionar um evento no seu *mural* do serviço *Facebook*

4.3 Resultados preliminares

Até ao momento da elaboração deste artigo, eram poucos os resultados disponíveis. No entanto, são apresentados nesta secção alguns resultados preliminares já processados.

O teste com mais resultados disponíveis é o segundo teste, e destes resultados pode-se concluir que a Janela de Divergência varia entre aproximadamente $1s$ e $4,5s$. Existem no entanto picos que podem levar a que a Janela de Divergência tome valores superiores a $8s$. Tal pode observado na Figura 1(a), que apresenta o histograma do tempo médio entre a realização de uma operação de escrita e os agentes observarem os seus efeitos. A Figura 1(b) apresenta a mesma informação, mas dividindo esta Janela de Divergência por várias localizações testadas. Podemos observar que a maioria das operações de leitura consegue observar o resultado das operações de escrita cerca de $2s$ após estas terem retornado.

Um dos objectivos secundários deste teste seria observar se a Janela de Divergência era idêntica tanto para um agente que estivesse na mesma região do agente de escrita como para um agente que estivesse numa região diferente. Os resultados indicam que esta diferença está presente: no caso em que o leitor



(a) Global (b) Por localização

Figura 1. Histograma da Janela de Divergência.

está na região onde foi executada a escrita, os valores tendem a ser inferiores aos valores de quando o leitor está numa região diferente.

Dos resultados obtidos também se pode observar que a propriedade *Read Your Writes* aparenta ser preservada: assim que uma escrita retorna, os efeitos desta são observáveis pelo agente que efectuou a escrita. Outra propriedade que aparenta ser preservada é *Monotonic Reads*. Os resultados mostram que após um agente observar os efeitos de uma operação de escrita, este não observa uma versão dos dados onde os efeitos da escrita não são observáveis.

5 Trabalho relacionado

Uma das ferramentas mais usadas na analise de sistemas replicados é *Yahoo! Cloud Serving Benchmark* [7]. Esta ferramenta permite o estudo e posterior comparação de performance de diferentes serviços, permitindo ainda a criação de extensões como a desenvolvida por Patil et al. [8], que possibilita a analise de consistência.

Alguns estudos à consistência de diferentes sistemas incluem os estudos realizados por Wada et al. [5] e Bermbach et al. [6]. Apesar de ambos se focarem na analise de consistência de serviços de armazenamento de dados, estes têm algumas diferenças entre si nos sistemas estudados e na metodologia dos testes. No estudo realizado por Wada et al. foram analisados os sistemas Amazon SimpleDB, Amazon S3, Google App Engine, e finalmente Microsoft Azure Table e Blob Storage. Estes sistemas foram testados sob diferentes configurações (e.g., escritor e leitor na mesma *thread*, em diferentes processos e ainda em diferentes máquinas virtuais geograficamente separadas). No caso do sistema Amazon SimpleDB foram ainda testadas as propriedades *Read Your Writes*, *Monotonic Reads* e *Monotonic Writes*. O estudo feito por Bermbach et al. focou-se em apenas num sistema, Amazon S3, mas ao contrário do estudo anterior, a metodologia deste inclui vários agentes a efectuar operações de escrita concorrentemente.

Alguns estudos mais analíticos tendem a focar-se no desenvolvimento de um modelo que permita a análise de consistência de um dado sistema sem ser necessário testá-lo directamente. Um destes modelos foi elaborado por Anderson et al. [11], baseando-se nas semânticas de consistência propostas por Lamport [12] e mais tarde complementadas no trabalho desenvolvido por Pierce e Alvisi [13]. Este modelo permite a analise da consistência de *key-value*

stores através construção de grafos conforme as operações executadas, permitindo detectar assim violações das semânticas de consistência de acordo com algumas regras definidas pelos autores.

Um outro modelo para analise de consistência de sistemas com base em grafos é proposto por Karmal et al. [10]. Neste estudo é construido um grafo conforme as operações executadas, sendo este uma aproximação do grafo global de dependências. A ocorrência de ciclos no grafo representa uma possível violação de consistência, sendo que por isso o número de ciclos é proporcional ao número de violações.

Outros estudos com uma vertente mais analítica incluem os realizados por Bailis et al. [9], que permite limitar a inconsistência de sistemas replicados que recorram a *quóruns* fracos, e ainda o modelo desenvolvido por Yu e Vahdat [14] que tem como objectivo impor um limite no desvio máximo entre o estado local de uma réplica e um estado final consistente que todas as réplicas deveriam ter. Este desvio pode ser medido e limitado segundo três métricas de forma a conseguir ser aplicado sobre diferentes tipos de dados, adequando-se assim a diferentes casos de uso.

6 Conclusões e Trabalho Futuro

Neste artigo apresentamos uma nova metodologia para o estudo e análise de garantias de consistência de serviços geo-distribuídos. Estudámos a aplicabilidade da nossa metodologia através da aplicação de um protótipo desenvolvido à rede social Google+. Esta metodologia pode, no entanto, ser aplicado a qualquer serviço, visto apenas usar a interface pública desse serviço sem necessidade de conhecimento a-priori sobre o funcionamento interno do mesmo.

Apresentámos resultados preliminares deste teste piloto, os quais permitiram tecer algumas considerações sobre as propriedade do modelo de consistência oferecidas pelo serviço estudado. Nomeadamente verificou-se que, aparentemente, as garantias de *Read Your Writes* e *Monotonic Reads* são efectivamente garantidas. Observou-se também que neste serviço a Janela de Divergência apresentava valores variaveis, podendo chegar a $9ms$, indicando que o modelo de consistência oferecido não corresponde a linearizibilidade.

Futuramente, iremos estender o nosso teste piloto, e também estender esta metodologia de forma a considerar outras garantias relevantes. Adicionalmente iremos estabelecer testes padrão, baseados nas propriedades relevantes, para validar de forma automática implementações que pretendem oferecer modelos de consistência específicos.

Referências

1. Terry, D. B., Demers, A. J., Peterson, Karin, Spreitzer, M. J., Theimer, M. M., Welch, B. B.: Session Guarantees for Weakly Consistent Replicated Data. In: Proc. of the 3rd International Conference on Parallel and Distributed Information Systems, pp. 140–149. IEEE Computer Society, Washington D. C. (1994)
2. S. Gilbert and N. Lynch.: Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. In: SIGACT News 33.2, pp. 51–-59. (2002)
3. E. A. Brewer.: Towards robust distributed systems (abstract). In: Proc. of the nineteenth annual ACM PODC. Portland, Oregon, USA. (2000)
4. D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in Bayou, a weakly connected replicated storage system, pp. 172-–182. In: Proc. of the fifteenth ACM SOSP '95. Copper Mountain, Colorado, USA. (1995)
5. H. Wada, A. Fekete, L. Zhao, K. Lee, and A. Liu. Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers' Perspective, pp. 134-–143. In: CIDR. (2011)
6. D. Bermbach and S. Tai. Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior, pp. 1:1-–1:6. In: Proc. of the 6th Workshop MW4SOC '11. Lisbon, Portugal. (2011)
7. B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with YCSB, pp. 143–154. In: Proc. of the 1st ACM SoCC '10. Indianapolis, Indiana, USA. (2010)
8. S. Patil, M. Polte, K. Ren, W. Tantisiriroj, L. Xiao, J. López, G. Gibson, A. Fuchs, and B. Rinaldi. YCSB++: benchmarking and performance debugging advanced features in scalable table stores, pp. 9:1–9:14. In: Proc. of the 2nd ACM SOCC '11. Cascais, Portugal. (2011)
9. P. Bailis, S. Venkataraman, M. J. Franklin, J. M. Hellerstein, and I. Stoica. Probabilistically bounded staleness for practical partial quorums, pp. 776—787. In: Proc. VLDB Endow. 5.8 (2012).
10. K. Zellag and B. Kemme. How consistent is your cloud application?, pp. 6:1—6:14. In: Proc. of the 3rd ACM Symposium on Cloud Computing. SoCC '12. San Jose, California. (2012)
11. E. Anderson, X. Li, M. A. Shah, J. Tucek, and J. J. Wylie. What consistency does your key-value store actually provide?, pp. 1-–16 In: Proc. of the Sixth HotDep'10. Vancouver, BC, Canada: USENIX Association. (2010)
12. L. Lamport. On interprocess communication, Part I: Basic formalism and Part II: Algorithms, pp. 77–101 In: Distributed Computing 1, 2. June 1986
13. E. Pierce and L Alvisi. A recipe for atomic semantics for Byzantine quorum systems. Tech. rep. University of Texas at Austin, Department of Computer Sciences. (2000)
14. H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services, pp. 239—282. In: ACM Trans. Comput. Syst. 20.3. (Aug. 2002)
15. Y. Saito and M. Shapiro. Optimistic replication. ACMComputing Surveys, 37(1). (2005)