Banco de Dados em Nuvem: Um Modelo para Garantia de Consistência dos Dados

Elyda Laisa S. X. Freitas, Fernando da Fonseca de Souza

Centro de Informática (CIn) - Universidade Federal de Pernambuco (UFPE)

Recife – Pernambuco – Brasil

{elsx,fdfd}@cin.ufpe.br

Nível: Mestrado

Ano de Ingresso no programa: 2012

Época esperada de conclusão: Março de 2014

Abstract. *This paper presents a work-in-progress focusing on a data consistency model, which uses the user's knowledge about the application to define which data needs strong consistency guarantees and which data does not need. Data with strong consistency guarantees are handled by eager update. In this case, an adapted architecture to cloud computing, that uses group communication, was designed. Data that do not require strong consistency can be treated with eventual consistency techniques.*

Resumo. *Este artigo apresenta o trabalho em andamento com foco em um modelo de consistência de dados, o qual se utiliza do conhecimento do usuário sobre a aplicação para definir quais dados necessitam de garantia de consistência forte e quais não. Os dados com garantia de consistência são tratados por meio da atualização ansiosa. Neste caso, uma arquitetura adaptada à nuvem, a qual se utiliza de comunicação em grupo, foi projetada. Para dados que não necessitam de consistência forte, técnicas de consistência eventual poderão ser utilizadas.*

Palavras-Chave

Banco de Dados como Serviço, Consistência de Dados, Banco de Dados em Nuvem.

1. Introdução e Motivação

Nos últimos anos, uma área de estudo de Tecnologia da Informação (TI) tem se destacado: a Computação em Nuvem, a qual tem por objetivo prover serviços de TI sob demanda aos usuários. O Sistema de Gerenciamento de Banco de Dados (SGBD) é um dos muitos serviços que podem ser disponibilizados na nuvem. Conhecido como DBaaS, do inglês *Database as a Service* (Banco de Dados como Serviço), esse novo paradigma tem recebido a atenção tanto da academia quanto de grandes empresas, como é o caso da Amazon, Oracle e Google (com os produtos Amazon S3, Oracle 12C e Bigtable, respectivamente).

Um dos problemas encontrados nos SGBD tradicionais foi "*a falta de suporte para particionamento dinâmico eficiente de dados, o que limitou a escalabilidade e a utilização de recursos*" (ELMASRI e NAVATHE, 2011).

No DBaaS, a distribuição, uma das características inerentes à nuvem, permite ao banco de dados armazenar grandes volumes de dados e crescer quase indefinidamente – aos limites da capacidade de armazenamento do *datacenter* hospedeiro. Por outro lado, essa característica impõe ao sistema de banco de dados algumas restrições. Conforme provado por Gilbert e Lynch (2002), não é possível atingir em um sistema distribuído, e ao mesmo tempo, as desejáveis características de Consistência, Disponibilidade e Tolerância à Partição (em caso de falha). Tal proposição é conhecida como Teorema CAP.

De acordo com Elmasri e Navathe (2011), para a preservação da consistência uma transação deve levar o banco de um estado consistente a outro. No entanto, diversos sistemas de bancos de dados em nuvem têm optado por relaxar a garantia de consistência, dando prioridade à disponibilidade do serviço (ZHAO et. al, 2012).

Para alguns tipos de aplicações, tal opção é perfeitamente válida, uma vez que não haja dados críticos armazenados. Por exemplo, não há maiores dificuldades se uma publicação não for imediatamente visualizada por todos os amigos de um usuário de uma rede social. No entanto, para grande parte das aplicações, inconsistências nos dados podem levar a transtornos imensuráveis, como no caso de uma aplicação bancária com valores equivocados. Desse modo, verifica-se que, em função do relaxamento da consistência, diversos sistemas de banco de dados em nuvem não estão aptos a receber todos os tipos de aplicações.

Entretanto, o relaxamento da consistência não é a única possibilidade: Dez anos após a concepção do Teorema CAP, Brewer (2012) afirma, em seu trabalho intitulado "*CAP Twelve Years Later: How the 'Rules' Have Changed*" (CAP Doze Anos Depois: Como as "Regras" Mudaram), que a compreensão desse teorema foi excessivamente simplificada, levando os projetistas a escolherem indiscriminadamente duas das proposições (como, por exemplo, tolerância à partição e disponibilidade; ou consistência e disponibilidade, apenas). Brewer (2012) recomenda, ainda, que a consistência não seja "cegamente" sacrificada.

Diante do exposto, torna-se perceptível a necessidade de explorar as diferentes nuances da consistência de um Banco de Dados em Nuvem, ao invés de simplesmente relaxar a consistência. Sendo assim, o seguinte problema de pesquisa foi formulado:

Como garantir a consistência dos dados armazenados em bancos de dados em nuvem a fim de permitir a utilização do modelo de DBaaS tanto por aplicações que necessitam de garantia de consistência quanto por aquelas que não necessitam?

2. Fundamentação Teórica

De acordo com a Oracle (2011), DBaaS "*É uma abordagem arquitetural e operacional que permite aos provedores de TI entregar funcionalidades de banco de dados como um serviço para um ou mais consumidores*". Isso significa dizer que no modelo DBaaS, o serviço oferecido pelo provedor são funcionalidades cotidianas dos SGBD, a saber: *backup* e recuperação de dados, entre outros. Um dos aspectos importantes na utilização do DBaaS é o modo como a consistência de dados é tratada. De acordo com Özsu e Valduriez (2011), "*Um banco de dados replicado está em um estado mutuamente consistente se todas as réplicas de cada um dos seus elementos de dados têm valores idênticos*".

Um método frequentemente utilizado para a distribuição nos sistemas de banco de dados em nuvem é a replicação, isto é, a cópia dos dados e armazenamento em mais de um local. Outra técnica de distribuição presente no DBaaS é o particionamento de dados. De acordo com Rahimi e Haug (2010), particionamento "*quebra uma tabela em dois ou mais pedaços chamados fragmentos ou partições e permite o armazenamento desses pedaços em diferentes locais*". Em geral, o particionamento de dados é realizado manualmente, sob responsabilidade do administrador do banco de dados. Na nuvem, o particionamento é dinâmico, realizado em tempo real.

Diversas abordagens podem ser utilizadas para replicar um banco de dados. Para que a replicação de dados ocorra de modo a garantir a consistência, pode-se utilizar a atualização ansiosa. Nessa abordagem, todas as atualizações são realizadas no contexto da transação. Consequentemente, quando a transação é confirmada, todas as réplicas possuem o mesmo valor (ÖZSU e VALDURIEZ, 2011).

Esse modelo de atualização geralmente utiliza a estratégia de *commit* em duas fases (*two-phase commit* - 2PC), que divide a execução do *commit* em duas etapas: na primeira, o coordenador envia um aviso de 'preparar para *commit*' a todos os servidores envolvidos na transação; na segunda fase, o coordenador pode enviar a mensagem 'realizar o *commit*' (caso tenha recebido todas as respostas positivas) ou abortar a transação. A atualização ansiosa, entretanto, pode causar problemas de desempenho, uma vez que durante a execução da transação, o dado a ser atualizado fica bloqueado (RAHIMI e HAUG, 2010).

Por outro lado, se não há necessidade de garantir a consistência dos dados de modo imediato, pode-se utilizar a abordagem de atualização preguiçosa. Nesse modelo, a atualização é realizada em uma réplica e repassada às outras réplicas de modo assíncrono. Como Gao e Diao (2010) afirmam: "*algoritmos de propagação preguiçosa postam as atualizações para as réplicas por meio de transações independentes, após a confirmação da transação de atualização no local de origem*". Deste modo, a atualização dos dados é um processo separado da propagação.

3. Caracterização da Contribuição

O presente trabalho parte do pressuposto de que uma mesma aplicação pode abrigar dados que necessitam de garantia de consistência e outros que não necessitam. Por exemplo, em um sistema de compra de passagens aéreas, a informação referente à quantidade de vagas disponíveis em determinado voo deve ser tratada de modo a garantir que todos os usuários visualizem a mesma informação, a fim de que não haja a venda de passagens excedentes. Nessa mesma aplicação, no entanto, a foto do cliente, parte do seu cadastro, não precisa ser imediatamente atualizada em todas as réplicas.

Para determinar o modo de tratamento dos dados, utiliza-se o conhecimento do usuário sobre sua aplicação: ele deve definir quais das tabelas devem ser tratadas com garantia de consistência e quais podem ter a garantia relaxada. No tocante aos dados com garantia de consistência, foi concebido um método diferente dos utilizados atualmente, conforme será mostrado na seção seguinte. Para isso, descartou-se o uso do protocolo 2PC, em benefício do desempenho.

Em seu lugar, há um modelo de comunicação em grupo, do tipo *publish-subscribe*, no qual "*assinantes (subscribers) registram seu interesse em um evento, ou um padrão de eventos, e são subsequentemente notificados, de forma assíncrona, de eventos gerados por aqueles que publicam (publishers)*" (EUGSTER et. al, 2003). No modelo proposto, após a definição de qual tabela deverá ser tratada com garantia de consistência, estas devem ser dispostas em réplicas da Camada 1, as quais se inscrevem para receber as atualizações desses dados, conforme mostra a Figura 1.

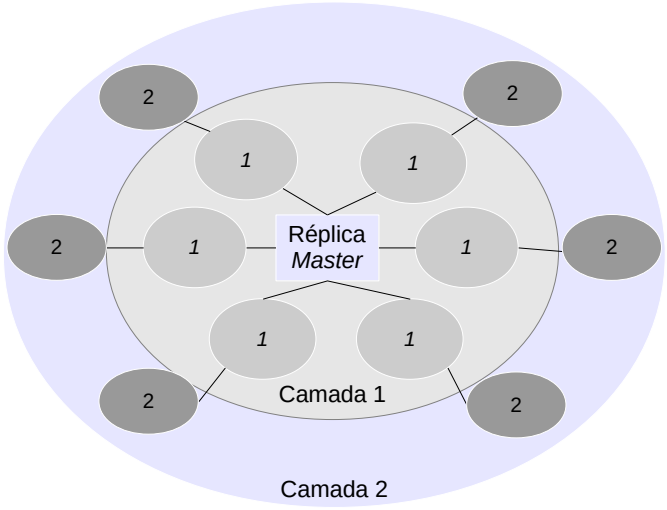

Figura 1. Modelo de replicação para garantia de consistência dos dados da Camada 1 e consistência eventual da Camada 2.

Conforme se pode verificar na imagem, há um único mestre, para o qual devem ser direcionadas as requisições. Quando a transação é confirmada no mestre, os *subscribers* são notificados e deverão aplicá-las em seus bancos de dados. Todos os *subscribers* deverão aplicar a atualização, a fim de que a consistência seja garantida. Uma vez que as atualizações podem ser realizadas em paralelo, espera-se uma melhoria de desempenho com relação aos modelos que se utilizam do *commit* em duas fases. Ademais, não há a utilização de bloqueios.

Para que não haja esperas indefinidas, devem ser estabelecidos prazos máximos de espera. Nestes casos, as réplicas não atualizadas deverão ficar indisponíveis para leitura até que sejam recuperadas e as atualizações aplicadas. Deste modo, mesmo que uma ou mais réplicas falhem e não sejam atualizadas, a consistência continua garantida. Além disso, consultas nas réplicas já atualizadas são imediatamente liberadas.

Para dar suporte à característica de particionamento dinâmico de dados, a seguinte regra deverá ser seguida: dados consistentes deverão ser particionados apenas em réplicas gerenciadas pela réplica *master,* de modo a garantir a consistência de dados. Ademais, todas as transações de leitura nas quais houver dados marcados com necessidade de consistência devem ser direcionadas às réplicas gerenciadas pelo *master* (Camada 1).

Em suma, a réplica *master* tem as seguintes funções: (1) receber as transações e aplicá-las normalmente; (2) capturar as transações aplicadas e preparar a mensagem; (3) armazenar as mensagens; e (4) gerenciar as réplicas e suas subscrições aos tópicos de interesse. Os tópicos são estruturas de dados criadas para armazenar as mensagens.

O *Provider* é o responsável por capturar as transações já confirmadas. Tal ação é realizada a partir do log da réplica *master*. Observe-se que a confirmação da transação na réplica *master* garante a verificação de restrições de consistência e eventuais dependências que possam existir no banco de dados, uma vez que não há mudanças nas funções-padrão do SGBD utilizado. Desse modo, essas transações podem ser executadas nas outras réplicas, mantendo tais benefícios, ao seguir a ordenação total.

Após capturar a transação, o *middleware* da réplica *master* deverá verificar o destino da mesma. Caso a mensagem seja destinada a uma réplica com dados que não necessitam de garantia de consistência, esta deverá apenas ser direcionada para as réplicas da Camada 2. Caso o destino seja as réplicas com dados que necessitam de garantia de consistência, a mensagem deverá ser armazenada no tópico.

No modelo *publish/subscribe* cada tópico possui um assunto e as réplicas subscrevem-se nos tópicos de acordo com o assunto de seu interesse (EUGSTER et. al, 2003). No modelo proposto, cada tópico deverá conter dados de uma tabela específica. Por exemplo, o tópico denominado “tópicoX” deverá conter todas as mensagens destinadas à tabela “passageiros”. As réplicas da Camada 1 deverão, portanto, subscrever-se nos tópicos com dados de suas respectivas tabelas. O responsável por coordenar os tópicos e suas particularidades é o TopicManager.

Em resumo, os elementos da arquitetura interagem do seguinte modo: 1) O cliente envia as mensagens (transações); 2) A réplica *master* recebe e aplica normalmente as transações; 3) O *Provider*, então, captura as mensagens e as organiza, adicionando dados como o destino, tempo de chegada e outras, caso sejam destinadas à Camada 1; ou as redireciona à Camada 2; 4) Através do *TopicManager*, o *Provider*, armazena a mensagem recebida no devido tópico; 5) O *TopicManager* devolve ao *Provider* a informação sobre qual o tópico onde a mensagem foi armazenada; e 6) Uma vez que a chegada das mensagens nos tópicos está em constante monitoramento por parte dos *subscribers*, estes deverão, neste momento, retirar a mensagem recebida.

Os dados que não necessitam de garantia de consistência (Camada 2), por sua vez, poderão ser tratados por meio da consistência eventual, que garante apenas que os

dados serão tornados consistentes eventualmente (ISLAM et. al, 2012). Nesse caso, técnicas consagradas poderão ser utilizadas sem alterações, uma vez que as mesmas já são largamente utilizadas no ambiente em nuvem. O trabalho de Vogels (2009) mostra diferentes variações da consistência eventual, as quais podem ser utilizadas no presente modelo. Para validar o modelo proposto, será desenvolvido um protótipo.

4. Trabalhos Relacionados

No que se refere à garantia de consistência em sistemas de banco de dados em nuvem, podem-se elencar diferentes trabalhos que tratam do referido tema. Um desses trabalhos é o de Wang et. al. (2010), o qual divide a estratégia de consistência em quatro categorias (que vai da consistência forte - isto é, o usuário sempre irá ler a versão mais atual do dado - à fraca, que não fornece essa garantia), de acordo com a frequência de leitura e escrita. Verifica-se, no entanto, que, ao contrário do presente trabalho, não há nenhuma participação do usuário no nível de consistência com o qual os dados devem ser tratados. Desse modo, este conhecimento sobre a aplicação é ignorado.

Outro trabalho relevante nesse contexto é o de Zhao et. al (2012), o qual apresenta um *framework* para replicação de banco de dados no ambiente em nuvem, onde o usuário deverá definir um SLA de atualidade dos dados para cada réplica. Deve-se observar, no entanto, que o modelo apresentado acima trata apenas da atualidade dos dados e não garante a consistência forte, uma vez que o SLA definido pelo usuário precisa ser violado para que o módulo de ação entre em atividade. Ademais, o processo de definição do SLA pode ser bastante dificultoso, já que não há nenhum suporte, por parte do sistema, a fim de auxiliar o usuário nesta tarefa.

Outro trabalho nesse contexto é o Harmony, apresentado no artigo de Chihoub et. al (2012). O Harmony tem por objetivo ajustar o nível de consistência adaptativamente de acordo com os requisitos da aplicação definidos pelo usuário (de 0 a 100% de taxa de leitura obsoleta aceitável) e do estado do sistema de armazenamento. Testes apresentados por Chihoub et. al (2012) mostram que o Harmony reduz a taxa de leitura de dados obsoletos, no entanto, o mesmo não garante a consistência forte dos dados.

Os trabalhos citados serviram de inspiração para este no modo como o usuário interfere diretamente no modelo, definindo os níveis de consistência desejados. No presente trabalho, o administrador de dados define as necessidades em um nível de granularidade mais baixo, que alcança a tabela na qual se deseja a garantia de consistência. Como o usuário conhece a aplicação, então a escolha das tabelas que deverão garantir a consistência torna-se uma tarefa mais simples.

5. Avaliação dos Resultados e Estado Atual do Trabalho

Espera-se, ao final deste trabalho, contribuir com a academia mediante a utilização de abordagens de replicação e propagação de dados reconhecidas e largamente estudadas (atualização ansiosa e preguiçosa) em um ambiente diferente daquele proposto inicialmente pelas mesmas, realizando as devidas adaptações para a nuvem.

Adicionalmente, o modelo de consistência desenvolvido explora os diferentes níveis da consistência de dados, conforme proposto por Brewer (2012). Atualmente, a revisão de literatura foi finalizada. O SGBD escolhido para o desenvolvimento do

protótipo foi o Oracle 12C, o qual permite o *download* do sistema, facilitando a montagem do ambiente controlado a ser utilizado para os testes. Por fim, será realizada uma pesquisa qualitativa, a fim de avaliar a adequação do protótipo desenvolvido.

Referências

Brewer, E. (2012) "CAP Twelve Years Later: How the 'Rules' Have Changed". Computer, vol.45, no.2, p.23-29, Feb. 2012.

Chihoub, H., Ibrahim, S., Antoniu, G. and Pérez, M. S. (2012) "Harmony: Towards Automated Self-Adaptive Consistency in Cloud Storage". In: 2012 IEEE International Conference on Cluster Computing (CLUSTER), p.293-301, 24-28 Sept. 2012.

Elmasri, Ramez and Navathe, Shamkant B. (2011) "Sistemas de Banco de Dados". 6ª Edição. Pearson Addison Wesley. São Paulo, 2011

Eugster, P. Th., Felber, P. A., Guerraoui, R. and Kermarrec, A. (2003) "The many faces of publish/subscribe". In: ACM Comput. Surv. 35, 2 (June 2003), p. 114-131.

Gao, A. and Diao, L. (2010) "Lazy update propagation for data replication in cloud computing". In: 5th International Conference on Pervasive Computing and Applications (ICPCA), p.250-254, 1-3 Dec. 2010.

Gilbert, S. and Lynch, N. (2002) "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". SIGACT News, vol.33, no., p.51–59, 2002.

Islam, M.A., Vrbsky, S.V. and Hoque, M.A. (2012) "Performance analysis of a tree-based consistency approach for cloud databases". In: International Conference on Computing, Networking and Communications (ICNC), p.39-44, Jan. 30 2012-Feb. 2 2012.

Oracle. (2012) "Database as a Service: Reference Architecture – An Overview. An Oracle White Paper on Enterprise Architecture". September 2011. http://www.oracle.com/technetwork/topics/entarch/oes-refarch-dbaas-508111.pdf. June 2013.

Özsu, M. T. and Valduriez, P. (2011) "Principles of Distributed Database Systems". Springer; 3rd ed. 2011 edition (March 2, 2011).

Rahimi, S. K. and Haug, F. S. (2010) "Distributed Database Management Systems: A Practical Approach". Wiley-IEEE Computer Society Pr; 1 edition (August 2, 2010).

Vogels, W. (2009) "Eventually consistent". Commun. ACM, vol. 52, pp. 40–44, January 2009.

Wang, X., Yang, S., Wang, S., Niu, X. and Xu, J. (2010) "An Application-Based Adaptive Replica Consistency for Cloud Storage". 9th International Conference on Grid and Cooperative Computing (GCC), p.13-17, 1-5 Nov. 2010

Zhao, L., SAKR, S. and LIU, A. (2012) "Application-Managed Replication Controller for Cloud-Hosted Databases". In: IEEE Fifth International Conference on Cloud Computing (CLOUD), p. 922-929. Honolulu: 24-29 June 2012.