Replicação Multi-nível de Bases de Dados em Memória

Helder Martins, João Soares, João Lourenço, e Nuno Prequiça*

CITI — Departamento de Informática,
Universidade Nova de Lisboa, Portugal
hr.martins@campus.fct.unl.pt joao.soares@di.fct.unl.pt
{joao.lourenco,nuno.preguica}@fct.unl.pt

Resumo Os serviços Web são frequentemente suportados por sistemas com uma arquitetura em camadas, sendo utilizadas bases de dados relacionais para armazenamento dos dados. A replicação dos diversos componentes tem sido uma das formas utilizadas para obter melhorarias de escalabilidade destes serviços. Adicionalmente, a utilização de bases de dados em memória permite alcançar um desempenho mais elevado. No entanto é conhecida a fraca escalabilidade das bases de dados com o número de núcleos em máquinas multi-núcleo. Neste artigo propomos uma nova abordagem para lidar com este problema, intitulada *MacroDDB*. Utilizando uma solução de replicação hierárquica, a nossa proposta, replica a base da dados em vários nós, sendo que cada nó, por sua vez, executa um conjunto de réplicas da base de dados. Esta abordagem permite assim lidar com a falta de escalabilidade das bases de dados relacionais em máquinas multi-núcleo, o que por sua vez melhora a escalabilidade geral dos serviços.

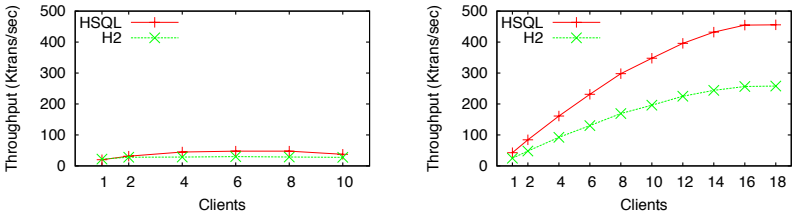
Keywords: Sistemas Distribuídos; Replicação; Macro-componente; Concorrência; Base de dados

1 Introdução

Os serviços Web são frequentemente suportados por sistemas baseados em arquiteturas multi-camada, sendo os dados destes serviços mantidos por sistemas de gestão de bases de dados (SGBD). De forma a melhorar a escalabilidade e desempenho dos serviços, mecanismos de replicação são normalmente utilizados ao nível dos componentes dos sistemas [13,1].

Mais recentemente, a utilização de bases de dados em memória permite a estes serviços alcançar melhorias de desempenho quando comparados a SGBD tradicionais [2,7]. No entanto, a fraca escalabilidade dos SGBD em ambientes multi-núcleo, inclusive dos suportados por memória, continua a ser um dos fatores limitativos do desempenho destes sistemas [14].

* Este trabalho foi parcialmente financiado pela Fundação para a Ciência e Tecnologia (FCT/MCTES), no contexto dos projetos de investigação PTDC/EIA-EIA/108963/2008 e PTDC/EIA-EIA/113613/2009.



(a) Um SGBD (b) Um SGBD por cliente

Figura 1. Escalabilidade para cargas de 100% leituras

Estudos mostram que os actuais SGBD consomem até 30% do seu tempo em operações relacionadas com sincronização (ex: *locking* e *latching*), até mesmo quando apenas um cliente (i.e., apenas uma *thread*) está a executar no sistema [5]. Adicionalmente, a execução concorrente de duas operações pode ser mais lenta que a execução sequencial das mesmas devido à interação das operações [18].

Escalabilidade de SGBD em memória Comparativamente a bases de dados suportadas por disco, as bases de dados em memória não incorrem em *overhead* nem em contenção no acesso a I/O. Desta forma, é expectável que estes sistemas escalem com o número de núcleos dos processadores atuais. De forma a verificar a escalabilidade dos sistemas atuais, corremos o benchmark TPC-C com uma carga de 100% de leituras nos sistemas HSQL e H2 numa Sun Fire x4600 com 16 núcleos e 32 Gbytes de RAM.

Os resultados, representados na figura 1(a), mostram a falta de escalabilidade destes motores, mesmo para cargas que não conflituam. De forma a melhor compreender se a falta de escalabilidade se deve a falta de recursos, corremos, concorrentemente, um número crescente de pares cliente e motor de bases de dados na mesma máquina. A figura 1(b) mostra dos resultados obtidos desta experiência, a qual representa o débito agregado. Estes resultados põem em evidência que o problema da falta de escalabilidade dos SGBD se deve ao seu desenho e não aos recursos disponíveis.

1.1 Abordagem proposta

Em [14] mostra-se as limitações à escalabilidade dos SGBD em máquinas multi-núcleos, inclusive em SGBD conhecidos como o *MySQL* e o *PostgreSQL*. Este trabalho apresenta o desenho do MacroDDB, uma solução baseada em replicação de bases de dados hierárquica, suportando a semântica *snapshot isolation*, na qual a base de dados é replicada por vários nós, sendo que cada nó, por sua vez, mantém um conjunto de réplicas da base de dados.

O MacroDDB tem uma organização que em cada nó existe uma réplica que é a primária e as outras são secundárias. Uma transação de leitura executa numa

réplica secundária, não levando a nenhuma troca de mensagens com as outras réplicas do mesmo nó ou de outros nós. As transações de escrita executam inicialmente na réplica primária do nó em que executa o cliente até ao momento da validação final. Neste momento, a transação é validada, verificando a existência de transações concorrentes a executar noutros nós que conflituam com a transação executada [4]. A solução é baseada na utilização dum sistema de comunicação em grupo, como proposto por Kemmet et. al.[10]. Após a validação da transação, as restantes réplicas locais são atualizadas assincronamente.

Esta aproximação hierárquica permite minimizar as mensagens trocadas na rede e o número de réplicas em que a validação das transações necessita de ser executada. A solução proposta neste artigo baseia-se em SGBD suportados em memória.

2 MacroDB: Bases de Dados Replicadas numa Máquina

Tal como posto em evidência anteriormente, as atuais arquiteturas dos SGBDs apresentam uma fraca escalabilidade em máquinas multi-núcleo [14]. Isto deve-se ao facto dos SGBDs se basearem numa arquitetura otimizada para acessos ao disco rígido.

A nossa solução para este problema passa por considerar cada nó do sistema como um sistemas distribuído de baixa latência com memória partilhada. Desta forma, e utilizando técnicas de replicação, a nossa solução, para cada nó, passa pela utilização de uma nova abstração, intitulada *Macro-Componente*. Esta abstração encapsula várias implementações de uma mesma interface, neste caso base de dados, num único componente, tal como descrito a seguir.

MacroDB [15] é um exemplo de um Macro-Componente [11] desenhado para melhorar a performance de SGBDs em memória em processadores multi-núcleo. Para isso, replica a base da dados em diferentes motores de SGBDs, executando todos no mesmo nó, oferecendo uma visão global consistente aos clientes.

MacroDB utiliza uma estratégia de replicação *primário-secundário* [17,6]. Transações de escrita, efetuadas pelo clientes, executam concorrentemente na réplica primária, sendo propagadas assincronamente e em *batch*, para as réplicas secundárias após *commit* no primário. As transações de leituras executam concorrentemente nas réplicas secundárias.

Esta estratégia possibilita que transações de leitura não abortem transações de escrita, visto executarem em réplicas distintas. A execução de transações de escrita, nas réplicas secundárias, como uma operação única, reduz o tempo de aquisição de *locks*, minimizando assim a interferência entre transações. Para além disso, transações de leitura concorrentes são reencaminhadas para réplicas secundárias distintas, possibilitando assim balanceamento da carga pelas diferentes réplicas e melhorias de desempenho consideráveis [15].

De forma a oferecer uma visão global consistente da base de dados, o *runtime* do sistema garante a execução das transações de escrita nas réplicas secundárias pela sua ordem correta, i.e., pela mesma ordem em que estas fizeram *commit* na réplica primária, garantindo assim a consistência do estado das diferentes

réplicas do sistema. Para além disso, transações de leitura podem ainda ser temporariamente atrasadas de forma a garantir que a réplica na qual executam se encontre atualizada, i. e., que tenha executado todas as transações de escrita que fizeram *commit* posteriormente ao início da transação que irá executar.

3 MacroDDB: Macro-Componentes Replicados e Distribuídos

O MacroDDB estende o MacroDB para um contexto distribuído, permitindo a replicação dum MacroDB por múltiplos nós dum centro de dados. Nesta secção descreve-se o desenho e implementação do MacroDDB.

3.1 Arquitetura do Sistema

Um MacroDDB é um sistema de replicação de bases de dados, organizado como se apresenta na Figura 2. O MacroDDB é composto pelos seguintes componentes a executar em cada nó: (i) um MacroDB composto por múltiplas réplicas da base de dados; (ii) a componente de replicação, responsável pela coordenação da execução das operações nos múltiplos nós; e (iii) a componente do cliente, responsável pela interação da aplicação com o sistema.

Uma aplicação interage com o sistema usando uma interface de bases de dados standard - JDBC na nossa implementação. Desta forma, para uma aplicação, o MacroDDB responde como uma qualquer base de dados. As operações executadas pela aplicação são processados inicialmente pela componente do cliente, a qual cria uma representação da operação. Estas operações são depois processadas pela camada de replicação que executa a operação no MacroDB local ou se coordena com as componentes de replicação dos outros nós para garantir a execução coordenada das operações. Esta coordenação é efetuada recorrendo a um sistema de comunicação em grupo. As operações são finalmente executadas no MacroDB local, onde o processamento segue a aproximação discutida na Secção 2. Nas secções seguintes descreve-se em detalhe os vários componentes do sistema e o modelo de execução das transações.

Na solução atual, um MacroDDB replica completamente a base de dados. Esta aproximação, apesar de apresentar inconvenientes relativos ao espaço ocupado, tem uma vantagem importante: todas as transações de leitura podem ser executadas localmente e num único nó.

3.2 Componente do Cliente

A componente do cliente (*Cliente* na Figura 2) é o componente do MacroDDB que a aplicação cliente utiliza para aceder às funcionalidades oferecidas pelo sistema, expondo a mesma interface do MacroDB, ou seja, a interface JDBC. Esta componente permite esconder à aplicação a localização e organização das réplicas das bases de dados. Para cada operação executada, cria-se uma representação da operação, a que chamamos macro-operação, que será usada para executar a

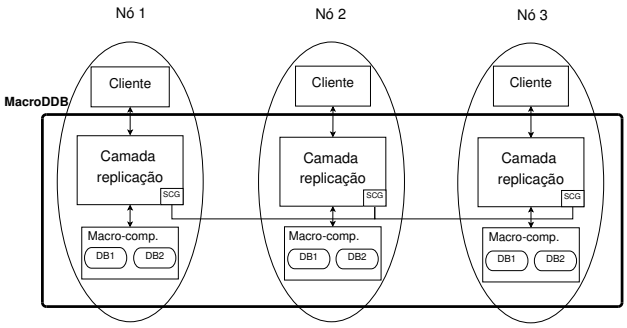


Figura 2. Arquitetura do Macro-componente distribuído

operação nos macro-componentes. A operação é propagada para a componente de replicação do MacroDDB à qual a componente cliente está ligada.

Macro-operação A Macro-operação contém a definição de uma dada operação ou seja, a sua implementação concreta. Para cada operação da interface é necessário existir uma definição da operação em forma de Macro-operação. Para isso a macro-operação apresenta funções que são invocadas posteriormente na componente de replicação, que contêm a implementação da operação.

3.3 Componente de Replicação

A componente de replicação (ou *Camada de replicação* na Figura 2) é responsável pela execução das operações recebidas das aplicações e de outras réplicas do MacroDDB no MacroDB local. A componente de replicação está organizada internamente nos seguintes módulos: *a*) sistema de comunicação com o cliente; *b*) sistema de comunicação em grupo; *c*) distribuidor; e o *d*) tradutor.

Sistema de comunicação com cliente O sistema de comunicação com o cliente é responsável pela interação com o cliente. Quando se recebe uma operação do cliente, esta é colocada numa fila de mensagens relativa ao cliente. Após a mensagem ser processada pelo componente módulo distribuidor, o resultado da execução da operação é devolvido ao cliente.

Sistema de comunicação em grupo O sistema de comunicação em grupo gere a comunicação com as outras réplicas do MacroDDB, sendo responsável pelo envio e receção de mensagens. Este módulo usa o sistema de comunicação em grupo JGroups para efetuar a comunicação.

Quando chega uma mensagem, a mensagem é inserida numa fila de pedidos não processados, permitindo libertar os recursos do sistema de comunicação em grupo e manter informação sobre a ordem das operações. Quando é solicitado o envio duma mensagem, recorre-se diretamente às operações definidas no sistema de comunicação em grupo usado.

Distribuidor O distribuidor é o módulo responsável pela execução das operações, consumindo as mensagens colocadas na fila de cada cliente e na fila do sistema de comunicação em grupo. O distribuidor tem a responsabilidade de determinar como cada operação deve ser executada e quando é que pode ser executada. Para isso as operações contém informação, que é utilizada pelo distribuidor que determina que operações podem ser executadas em simultâneo. Para permitir a execução concorrente de múltiplas operações, quando isso é aceitável, o distribuidor tem vários fios de execução a consumir as operações. Em cada momento existe, no máximo, um fio de execução a processar operações de cada fila.

Tradutor O tradutor é responsável por receber o pedido (ou operação) que é enviada pelo componente distribuidor. Para esse efeito o tradutor oferece a função *makeCall* que executa a operação definida na Macro-operação que foi criada na componente do cliente.

A execução da operação definida na Macro-operação pode retornar três tipos de resposta: (i) resposta vazia para operações que não devolvem resultado; (ii) resposta com conteúdo; e (iii) resposta com exceção, quando uma operação termina em erro. Caso se trate duma operação enviada diretamente por um cliente, o resultado é devolvido ao cliente.

Isolar a implementação das operações dentro de uma só função traz vantagens. Por um lado a implementação está isolada e portanto torna-se mais simples encontrar *bugs* nas implementações. Adicionalmente nem todo o conteúdo retornado pelo MacroDB é possível de ser enviado pela rede sem tratamento, portanto podemos efetuar computação arbitrária de forma a tornar o conteúdo viável para envio na rede.

3.4 Execução das transações

Tendo apresentado as componentes que constituem o MacroDDB, nesta secção detalha-se a execução das transações. Como foi descrito, as operações executadas pela aplicação são enviadas pela componente do cliente para a componente de replicação, que define a lógica de processamento das operações.

Para cada transação iniciada, todas as operações da mesma são executadas inicialmente no MacroDB local - esta execução pode ocorrer na réplica primária do MacroDB caso seja uma transação de escrita ou numa réplica secundária caso se trate duma transação assinalada como transação de leitura.

No momento do *commit*, o processamento das transações difere. No caso das transações de leitura, a função *commit* do MacroDB é invocada e a transação conclui-se com sucesso (assumindo que o motor de base de dados garante que uma transação de leitura não pode falhar no momento do *commit*).

No caso de uma transação de escrita é necessário validar a execução da transação. Para tal, verificam-se os potenciais conflitos escrita-escrita com transações que executaram concorrentemente. Esta verificação é efetuada recorrendo à propagação dos conjuntos de escrita através dum sistema de comunicação em grupo,

usando *multicast* atómico. Esta aproximação, usada anteriormente em vários sistemas [9,4], garante que os conjuntos de escrita são recebidos e processados pela mesma ordem em todas as réplicas.

4 Trabalho Relacionado

A performance dos serviço Web está diretamente relacionada com a performance dos seus componentes, em particular do SGBDs. Replicação é um dos mecanismos mais utilizadas para melhorar a performance e a disponibilidade de SGBDs [16,3,12,1]. Diferentes propostas sugerem a utilização de *middleware* para melhorar a performance SGBD em ambientes distribuídos [13,16,3,12]. Estes trabalhos são ortogonais ao nosso, visto não considerarem a utilização de recursos da cada nó na performance geral do sistema. Ou seja, apenas consideram a performance do agregado dos nós do sistema, e não a utilização dos recursos disponíveis em cada nó.

Ao tratar cada nó do sistema como um sistema distribuído de baixa latência, estendido com memória partilhada, MacroDDB apresenta uma solução de replicação hierárquica capaz de melhorar a performance global do sistema, garantindo o eficiente aproveitamento dos recursos de cada nó.

Visto que o MacroDDB integra o MacroDB, cujo o objetivo é tirar o devido partido dos recursos oferecidos pelos sistemas multi-núcleo, naturalmente o MacroDDB torna-se uma solução para tirar partido de um conjunto de nós de máquinas multi-núcleo. O sistema Eve partilha os objetivos do nosso sistema [8]. Nessa solução, os pedidos são enviados para todas as réplicas, onde posteriormente os mesmos são agrupados em conjuntos independentes e executados concorrentemente de uma forma otimista. No caso da previsão estar errada e o valor das réplicas nos diferentes nós divergir, o sistema precisa de passar por um passo de desfazer as alterações e aplicar as mesmas sequencialmente. Na nossa solução garantimos que os nós terminam no mesmo estado, visto que são obrigados a executar os pedidos pela ordem total dada pela difusão atómica.

5 Conclusão

Neste artigo apresentamos o MacroDDB, uma solução de replicação hierárquica de bases de dados, que oferece uma semâtica de *snapshot isolation*.

Contrariamente a abordagens convencionais, MacroDDB mantém, em cada nó do sistema, um conjunto de réplicas da bases de dados. Estas utilizam um esquema de primário-secundário, em que transações de escrita executam na réplica primária e as de leitura em réplicas secundárias. Desta forma, transações de escrita podem executar localmente e num determinado nó sem que para isso haja necessidade de coordenação inicial com os restantes nós do sistemas. Apenas na validação de uma transacção de escrita é verificada a existência de transações concorrentes, que executem noutras máquinas, que conflituam, utilizando para isso comunicação atómica em grupo. Após validação, as restantes réplicas locais

são atualizadas assincronamente. Transações de leitura executam nas réplicas secundárias sem qualquer necessidade de coordenação com as restantes máquinas do sistemas.

Esta abordagem permite assim lidar com a falta de escalabilidade dos sistemas de gestão de bases de dados em multi-núcleos. Adicionalmente possibilita minimizar a trocas de mensagens entre os diferentes máquinas do sistema.

Referências

1. Jason Baker, Chris Bond, James Corbett, J. J. Furman, Andrey Khorlin, James Larson, Jean-Michel Leon, Yawei Li, Alexander Lloyd, and Vadim Yushprakh. Megastore: Providing scalable, highly available storage for interactive services. In *CIDR*, pages 223–234. www.cidrdb.org, 2011.
2. Lásaro Camargos, Fernando Pedone, and Marcin Wieloch. Sprint: a middleware for high-performance transaction processing. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 385–398, New York, NY, USA, 2007. ACM.
3. Sameh Elnikety, Steven Dropsho, and Fernando Pedone. Tashkent: uniting durability with transaction ordering for high-performance scalable database replication. In *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*, EuroSys '06, pages 117–130, New York, NY, USA, 2006. ACM.
4. Sameh Elnikety, Willy Zwaenepoel, and Fernando Pedone. Database replication using generalized snapshot isolation. In *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, SRDS '05, pages 73–84, Washington, DC, USA, 2005. IEEE Computer Society.
5. Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, and Michael Stonebraker. Oltp through the looking glass, and what we found there. In *Proceedings of the 2008 ACM SIGMOD-ICMD*, SIGMOD '08, New York, NY, USA, 2008. ACM.
6. Abdelsalam A. Helal, Bharat K. Bhargava, and Abdelsalam A. Heddaya. *Replication Techniques in Distributed Systems*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.
7. Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, and Daniel J. Abadi. H-store: a high-performance, distributed main memory transaction processing system. *Proc. VLDB Endow.*, 1(2):1496–1499, August 2008.
8. Manos Kapritsos, Yang Wang, Vivien Quema, Allen Clement, Lorenzo Alvisi, and Mike Dahlin. All about eve: execute-verify replication for multi-core servers. In *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation*, OSDI'12, pages 237–250, Berkeley, CA, USA, 2012. USENIX Association.
9. Bettina Kemme and Gustavo Alonso. A suite of database replication protocols based on group communication primitives. In *Proceedings of the The 18th International Conference on Distributed Computing Systems*, ICDCS '98, pages 156–, Washington, DC, USA, 1998. IEEE Computer Society.
10. Bettina Kemme and Gustavo Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Trans. Database Syst.*, 25(3):333–379, September 2000.

11. Paulo Mariano. Repcomp - replicated software components for improved performance. Master's thesis, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa, September 2010.
12. Takeshi Mishima and Hiroshi Nakamura. Pangea: an eager database replication middleware guaranteeing snapshot isolation without modification of database servers. *Proc. VLDB Endow.*, 2(1):1066–1077, August 2009.
13. Christian Plattner and Gustavo Alonso. Ganymed: scalable replication for transactional web applications. In *Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware*, Middleware '04, pages 155–174, New York, NY, USA, 2004. Springer-Verlag New York, Inc.
14. Tudor-Ioan Salomie, Ionut Emanuel Subasu, Jana Giceva, and Gustavo Alonso. Database engines on multicores, why parallelize when you can distribute? In *Proceedings of the sixth conference on Computer systems*, EuroSys '11, pages 17–30, New York, NY, USA, 2011. ACM.
15. João Soares, João Lourenço, and Nuno M. Preguiça. Macrodb: Scaling database engines on multicores. In *Proceedings of Euro-Par 2013*, 2013.
16. Matthias Wiesmann and Andre Schiper. Comparison of database replication techniques based on total order broadcast. *IEEE Trans. on Knowl. and Data Eng.*, 17(4):551–566, April 2005.
17. Matthias Wiesmann, André Schiper, Fernando Pedone, Bettina Kemme, and Gustavo Alonso. Database replication techniques: A three parameter classification. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, SRDS '00, pages 206–, Washington, DC, USA, 2000. IEEE Computer Society.
18. Jingren Zhou, John Cieslewicz, Kenneth A. Ross, and Mihir Shah. Improving database performance on simultaneous multithreading processors. In *Proceedings of the 31st VLDB*, VLDB '05. VLDB Endowment, 2005.