Replicação Parcial em Sistemas de Memória Transaccional *

Pedro Ruivo, Maria Couceiro, Paolo Romano, and Luís Rodrigues

{pruivo, mcouceiro, romanop}@gsd.inesc-id.pt, ler@ist.utl.pt
INESC-ID Lisboa, Instituto Superior Técnico, Universidade Técnica de Lisboa

Resumo Hoje em dia, as "caches" em memória distribuídas são cada vez mais utilizadas como forma de aumentar o desempenho de aplicações com acesso intensivo a grandes quantidades de dados, desacoplando o acesso a memória persistente do caminho crítico da aplicação. Este artigo propõe e avalia técnicas de replicação parcial dos dados de forma aumentar a fiabilidade deste tipo de sistemas e também suportar a execução concorrente de operações de leitura. A solução é avaliada recorrendo a um protótipo desenvolvido sobre o sistema de código aberto da RedHat, o *Infinispan*.

Abstract Nowadays, the distributed in-memory caches are increasingly used as a way to improve the performance of applications that require frequent access to large amounts of data, by decoupling the persistent memory access from the critical path of the application. This paper proposes and evaluates techniques for supporting partial replication of data to increase the reliability of such systems and also to support the concurrent execution of read operations in different nodes. The solution is evaluated with a prototype developed on top of *Infinispan*, a RedHat open source system.

Keywords: Replicação Parcial, Memória Distribuída, Memória Transaccional, Difusão Atómica

1 Introdução

Hoje em dia, as "caches" em memória distribuídas são cada vez mais utilizadas como forma de aumentar o desempenho de aplicações com acesso intensivo a grandes quantidades de dados. O *Youtube*, *Wikipedia*, *Twitter*, *Facebook* são alguns exemplos, entre muitos, de aplicações que fazem uso deste tipo de tecnologia.

Uma das principais motivações para a utilização de caches distribuídas em memória é o aumento do desempenho das aplicações, uma vez que o acesso a informação em memória, mesmo que numa máquina remota, é substancialmente

* Este trabalho foi parcialmente suportado pela FCT (financiamento multianual do INESC-ID) através dos fundos do Programa PIDDAC e através do projecto "ARISTOS" (PTDC/EIA-EIA/102496/2008) e pela Comissão Europeia através do projecto "Cloud-TM" (257784)

mais rápido do que o acesso ao disco. Estas caches não se limitam a suportar operações de leitura: as operações de escrita são também realizadas em memória, sendo a transferência para disco realizada nos bastidores, de forma diferida.

Para além disso, muitas destas aplicações não necessitam de usar SQL no acesso aos dados, conseguindo operar através de uma interface mais simples, como por exemplo a oferecida por uma tabela de dispersão, o que desencoraja a utilização de soluções baseadas em bases de dados tradicionais. No entanto, a possibilidade de realizar um conjunto de operações de forma atómica continua a ser útil em muitos padrões de utilização, pelo que as caches distribuídas mais recentes oferecem algum suporte para a execução de *transacções em memória* [5,20].

Neste contexto, a replicação dos dados armazenados em cache e distribuídos pela memória de múltiplos nós possui duas grandes vantagens: por um lado, permite distribuir a carga das operações de leitura por várias réplicas; por outro lado, assegura a sobrevivência dos dados na eventualidade da falha de uma das réplicas. Este último aspecto é de particular relevância, atendendo ao facto de os dados serem primeiro armazenados em memória volátil (cujo conteúdo se perde no caso de falha) e tornados persistentes de forma assíncrona.

As vantagens acima referidas necessitam de ser pesadas contra os custos envolvidos na replicação, a saber: por um lado, as réplicas consomem memória, reduzindo a quantidade de informação que pode ser armazenada na cache, podendo obrigar a um acesso mais frequente ao disco; por outro, quanto maior o número de réplicas maior poderá ser o custo de manter as mesmas coerentes.

Desta forma, justifica-se a utilização de *replicação parcial*, isto é, uma configuração da cache em que cada item é replicado num subconjunto dos nós da cache, sendo que nenhum nó possui todo os dados. Este artigo estuda a aplicação de técnicas de replicação parcial neste contexto. Apesar destas técnicas já terem sido experimentadas em sistemas de bases de dados distribuídas, existem diferenças significativas na caracterização das cargas impostas ao sistema, e no tipo de processamento que é feito na sua execução, que justificam o nosso trabalho. Em particular, os sistemas de gestão de bases de dados são obrigados a executar diversas fases (processamento do SQL, persistência síncrona, etc.) que acarretam custos que não existem nos sistemas de cache. Assim, os custos de coordenação associados a replicação parcial são, em termos relativos, amplificados. Este trabalho pretende contribuir para aferir em que medida a replicação parcial é viável e eficaz neste contexto.

O artigo apresenta as seguintes contribuições: descreve um conjunto de algoritmos para a concretização de replicação parcial em sistemas distribuídos de cache; faz uma avaliação experimental desta solução com base numa adaptação do *Infinispan*, uma cache distribuída de código aberto da RedHat.

O resto do artigo está organizado no seguinte modo. Na Secção 2 descreve-se resumidamente o *Infinispan* e da forma como este gere a distribuição e a replicação. Na Secção 3 são apresentados os mecanismos de replicação parcial desenvolvidos para enriquecer o *Infinispan*. Na Secção 4 avalia-se o desempe-

nho do sistema resultante. A Secção 5 compara a nossa solução com o trabalho relacionado. Finalmente, a Secção 6 conclui este artigo.

2 O *Infinispan*

O *Infinispan* [2] é uma cache em memória distribuída e transaccional, desenvolvida e distribuída pela RedHat em código aberto. O modelo de programação oferecido é o de um mapa que mantém uma projecção entre chaves e valores, permitindo inserir, ler, actualizar e remover objectos de dados através de uma chave que lhes é associada pelo programador. Esta interface é uma extensão da interface `java.util.Map` do Java. Para além disso o *Infinispan* oferece suporte para transacções: uma sequência de operações sobre o mapa pode ser delimitada por métodos que identificam o início e fim da transacção. Uma transacção é iniciada invocando o método `begin()` e termina com a invocação do método `commit()` ou `rollback()`. Se o método `commit()` termina com sucesso, todas as operações de actualização executadas durante a transacção tornam-se visíveis no mapa; caso contrário, estas actualizações são descartadas.

Um dos modelos de coerência oferecido pelo *Infinispan* é o "*Read Committed*" [6]. De acordo com este modelo de coerência, uma transacção só pode ler valores que já tenham sido confirmados. No entanto não são dadas garantias adicionais, nomeadamente não se garante que a execução concorrente é equivalente a uma execução em série das mesmas transacções. No *Infinispan* isto é conseguido da seguinte forma: quando se escreve num objecto, é obtido um trinco de escrita (e a transacção bloqueia se o trinco já tiver sido detido por outra transacção). Este trinco é mantido até que a transacção seja confirmada ou cancelada. Quando um objecto é lido, é lida a ultima versão confirmada (o *Infinispan* usa uma técnica simplificada de manutenção de múltiplas versões).

O algoritmo descrito no parágrafo anterior é baseado na propagação de actualizações diferida, onde todas as actualizações só são propagadas para as restantes réplicas quando a transacção está pronta para ser confirmada.

Uma vez que os objectos estão distribuídos por vários nós, é necessário assegurar que as actualizações realizadas por uma transacção são aplicadas na totalidade (ou todas canceladas). No *Infinispan* isto é conseguido recorrendo a um protocolo de confirmação em duas fases clássico (*2 Phase Commit* [14]).

Para suportar a coordenação entre réplicas o *Infinispan* usa o sistema de comunicação em grupo *JGroups* [3]. Este sistema permite que os nós possuam uma vista coerente da filiação no grupo de nós que participam na cache e, desta forma, possam fazer uma atribuição determinista e coerente da responsabilidade de armazenamento de cada chave. No entanto, uma vez que uma transacção não acede necessariamente a todos os membros do grupo (apenas a um subconjunto destes, nomeadamente aqueles que possuem cópias dos dados acedidos pela transacção), não são usadas primitivas de difusão para a troca de mensagens no acesso aos dados. Pelo contrário, todas as mensagens trocadas pelo sistema para leitura e escrita, assim como para suportar o protocolo de confirmação, usam primitivas de comunicação ponto-a-ponto fiáveis. Isto abre a janela

para que duas transacções que acedam aos mesmos dados de forma concorrente obtenham os trincos por ordem diferente em diferentes réplicas gerando uma situação de interbloqueio. Estas situações são resolvidas recorrendo a temporizadores, sendo particularmente perniciosas, pois os dados ficam indisponíveis por um longo período.

3 Replicação Parcial Eficiente

Como foi descrito na secção anterior, o *Infinispan* já possui, nativamente, suporte para replicação parcial. Infelizmente este suporte é bastante rudimentar: como é necessário realizar a aquisição de trincos em diversas réplicas, o sistema está sujeito a interbloqueios.

Neste artigo propõe-se um novo algoritmo para replicação parcial no *Infinispan*, com o intuito de obter ganhos no desempenho. O nosso algoritmo tira partido dos resultados recentes de investigação na área das bases de dados replicadas, fazendo um conjunto de adaptações e optimizações aos algoritmos anteriormente propostos, de forma a reduzir o custo da coordenação, uma vez que este custo é comparativamente maior nos sistemas de cache em memória do que nos sistemas de gestão de bases de dados, onde a execução das transacções é mais pesada.

3.1 Execução das Transacções

O nosso algoritmo baseia-se nos seguintes princípios. Cada transacção é executada localmente numa única máquina. A transacção mantém uma cópia local de todos os objectos lidos ou escritos pela transacção. Note-se que os dados estão distribuídos, pelo que a leitura de dados pode obrigar ao contacto de uma máquina remota. Este é um custo incontornável da distribuição. No entanto, como se descreveu anteriormente, as leituras nunca bloqueiam localmente, pelo que o acesso a dados remotos é limitado pelo tempo de ida-e-volta na rede. As operações de escrita não requerem interacção distribuída durante a execução da transacção.

Se a transacção é cancelada pelo utilizador, as actualizações são simplesmente descartadas, sem nunca serem aplicadas na cache nem se tornarem visíveis para outras transacções. Se a transacção estiver pronta a ser confirmada, as actualizações são enviadas numa única mensagem para todas as réplicas que necessitam de ser actualizadas. Este conjunto de réplicas é a união das réplicas de cada um dos objectos que foi actualizado pela transacção. Tipicamente, este conjunto é apenas um subconjunto do número de máquinas que fazem parte da cache.

As mensagens de difusão das actualizações acima referidas são enviadas por ordem total [12], isto é, se duas réplicas entregam duas actualizações, fazem-no exactamente na mesma ordem. Desta forma, as réplicas necessitam apenas de aplicar as actualizações recebidas pela ordem em que as receberam.

3.2 Concretização da Difusão Ordenada

O algoritmo anterior pressupõe a utilização de uma primitiva de difusão em grupo que assegure a ordem total entre múltiplos grupos. De facto, cada actualização é aplicada num conjunto de réplicas distinto, pelo que o conjunto de destinatários de diferentes mensagens em difusão pode ser diferente. Apesar disto, na intersecção de dois grupos distintos, as mensagem recebidas em comum devem ser totalmente ordenadas. Infelizmente, existem poucos sistemas de comunicação em grupo que suportam esta funcionalidade. Em particular, este tipo de garantias não é oferecido pelo *JGroups*.

Uma maneira simples de conseguir ordem total entre mensagens para grupos diferentes consiste em simular a difusão em grupo enviando todas as mensagens em ordem total para todas as máquinas (isto é, para um único super-grupo, que é a união de todos os sub-grupos possíveis). Posteriormente, cada máquina descarta as mensagens que não lhe são de facto destinadas. Este tipo de solução é designada por não-genuína [24], e é pouco eficiente do ponto de vista do desempenho, pois na prática todas as máquinas são envolvidas em todas as transacções, perdendo-se a capacidade de dispersão de carga proporcionada pelo particionamento dos dados por diferentes réplicas.

Desta forma, desenvolvemos também um protocolo de difusão selectiva com ordem total para o *JGroups*. Este protocolo é inspirado no protocolo utilizado na concretização original do sistema ISIS[21] e opera da seguinte maneira. Cada máquina possui um relógio lógico[18] que é incrementado quando recebe mensagens de DADOS ou mensagens de ORDEM. A difusão ordenada é iniciada enviando uma mensagem de DADOS para o grupo de réplicas participantes na transacção. Quando esta mensagem é recebida, cada réplica incrementa o seu relógio lógico, atribui essa estampilha temporal à mensagem, e coloca-a numa fila ordenada no estado *Pendente*; o número de ordem local é então enviado para o emissor da mensagem em difusão. O emissor, recolhe os vários números de sequência atribuídos em cada réplica, calcula o máximo destes valores, e envia este número para todas as replica numa mensagem de ORDEM. Ao receber uma mensagem de ordem, cada réplica actualiza o número da mensagem, reordenando-a na fila, caso seja necessário, marca a mensagem como *Final*, e actualiza o seu relógio lógico. Finalmente, as mensagens são entregues quando estão no estado *Final* e se encontram na cabeça da fila (isto é, não existe nenhuma outra mensagem, *Pendente* ou *Final* com um número inferior.

3.3 Discussão

O leitor atento poderá observar que o comportamento do protocolo de ordenação de mensagens acima descrito é bastante semelhante ao comportamento do protocolo de confirmação em duas fases nativo do *Infinispan*, nomeadamente no que se refere ao número de passos de comunicação. Desta forma, o nosso algoritmo possui o mesmo custo mas com uma grande vantagem: ao contrário do que acontece no algoritmo nativo, nossa solução não sofre de interbloqueios. Na

secção de avaliação que se segue, mostramos que esta característica pode acarretar ganhos de desempenho muito significativos para várias caracterizações da carga do sistema.

4 Avaliação

Nesta secção é feita uma avaliação experimental do algoritmo proposto para concretizar replicação parcial em sistemas de memória transaccional distribuída. Esta avaliação é feita com base num protótipo desenvolvido através da extensões ao código do *Infinispan* e do *JGroups*, de forma a concretizar o algoritmo anteriormente apresentado.

4.1 Bancada Experimental

Todas as experiências foram realizadas num *cluster* de 11 máquinas, onde cada máquina está equipada com dois processadores Quad-Core XEON a 2.0GHz e 8GB de memória RAM. O sistema operativo usado é o Linux 2.6.32.21 e as máquinas estão interligadas através de uma rede *Ethernet* privada com um débito de 1 Gigabit por segundo. As experiências usam um número variável de máquinas, entre 5 e 11, e duas configurações distintas do sistema *Infinispan*: a configuração nativa, baseada no protocolo com propagação diferida e coordenação através de confirmação em duas fase, e a configuração por nós desenvolvida, com base no algoritmo baseado em difusão em grupo com ordem total genuína. A configuração nativa do *Infinispan* tenta adquirir trincos com temporizadores configurados com o valor de 10 segundos e a detecção de interbloqueio encontra-se activada, cancelando uma das transacções num tempo inferior ao definido. Em ambas as configurações foi usado um grau de replicação de 2 e de 4, isto é, cada chave é armazenada em duas ou em quatro máquinas, respectivamente.

A carga no sistema, isto é as transacções que utilizam a cache, é gerada por uma aplicação de teste e avaliação de desempenho criada pela RedHat especificamente para este tipo de caches, denominada *Radargun*. Esta aplicação permite comparar o desempenho de várias caches distribuídas (tais como, *Infinispan*, *Ehcache* [1], *Coherence* [4], etc.), em diferentes cenários e, ao impor uma carga bastante elevada sobre os diferentes nós do sistema, permite-nos aferir qual o débito máximo de cada configuração.

Fizemos também algumas alterações a esta aplicação de teste, de forma a definir diferentes padrões de utilização através da configuração de diferentes taxa de conflitos (isto é, acessos concorrentes aos mesmos objectos). Desta forma, conseguimos um maior controlo sobre as experiências, estabelecendo mais facilmente correlações entre a taxa de conflitos, a ocorrência de situações de interbloqueio e o desempenho de ambas as configurações.

Em várias das experiências foi usado o seguinte padrão de carga. A aplicação tenta executar o maior número possível de transacções durante um período de 5 minutos, usando 8 fios de execução paralelos em cada máquina, que submetem concorrentemente transacções ao sistema. Cada transacção é constituída por 10

operações com uma média de 10% de operações de escrita. No entanto, garante-se que existe pelo menos uma operação de escrita por transacção, eliminando assim a existência de transacções só de leitura. Esta decisão foi efectuada pois as operações de leitura não envolvem sincronização entre as réplicas. Apesar de a taxa média de 10% de operações de escrita ser realista em muitos casos, é interessante avaliar e comparar o comportamento do sistema para taxas de escrita mais elevadas. Com esse objectivo, foi efectuada uma avaliação com as a taxa média de operação de escrita de 20%, 50% e 100%. No entanto, como só queremos verificar o comportamento dos protocolo com a variação das operações de escritas, esta avaliação foi efectuada com grau de replicação 2 (duas cópias por chave).

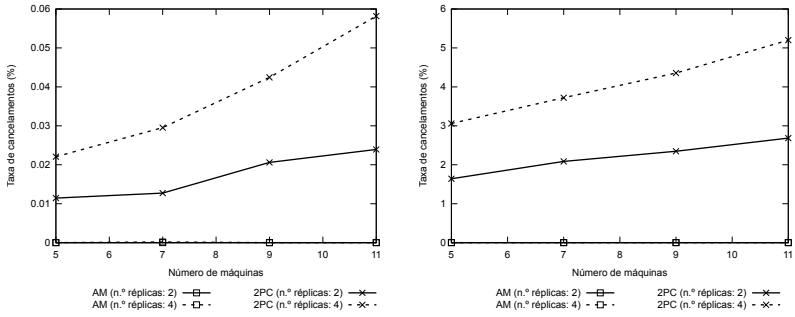
Finalmente, para simular cenários de baixa contenção, as transacções acedem a objectos aleatórios dentro de um conjunto de 100.000 chaves e para simular cenários de elevada contenção, as mesmas transacções acedem a um conjunto de apenas 1.000 chaves.

4.2 Resultados

Nos parágrafos seguintes fazemos uma avaliação comparativa da versão nativa do *Infinispan* (que aparece nos gráficos com a etiqueta "2PC") e a nossa solução baseada em difusão atómica (que aparece nos gráficos com a etiqueta "AM") usando três métricas de desempenho distintas: a taxa de cancelamentos, o débito do sistema (transacções confirmadas) e latência (duração da fase de confirmação).

Devido ao limite no número de páginas, só é apresentado valores referentes ao débito do sistema para a variação da taxa de operações de escrita. No entanto, são comentadas todas as métricas de desempenho.

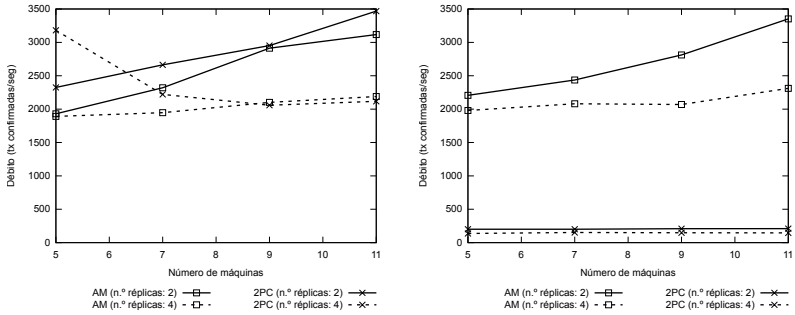
Taxa de Cancelamentos A Figura 1 mostra a taxa de cancelamentos dos dois algoritmos em cenários de baixa e elevada contenção, respectivamente. Como seria de esperar, devido à utilização de ordem total, a taxa de cancelamentos devido a interbloqueios é nula. Pelo contrário, na algoritmo nativo necessita de adquirir os trincos em todas as réplicas participantes, podendo, caso estes trincos sejam obtidos em diferentes ordens por transacções diferentes, gerar situações de interbloqueio que, por sua vez, geram cancelamentos. Este efeito é, também naturalmente, mais notório quando se aumenta o número de nós do sistema e quando se aumenta o número de cópias de cada item e, finalmente, em cenários de elevada contenção, onde a taxa de cancelamento chega aos 5% para um sistema de 11 nós com 4 cópias de cada item. As mesmas conclusão se aplicam quando se varia a taxa de operações de escrita, no entanto, a taxa de cancelamento para o 2PC sofre um aumento significativo, atingindo os 80% no cenário de elevada contenção, com 100% de operações de escrita e 11 nós. Este aumento é devido ao aumento da probabilidade de conflitos entre transacções originado pelo aumento das operações de escrita numa transacção.



(a) Baixa contenção (b) Elevada contenção

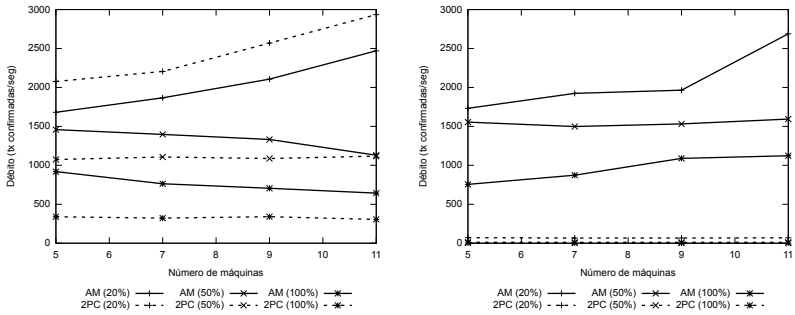
Figura 1: Taxa de cancelamentos

Débito A Figura 2 mostra o efeito da taxa de cancelamentos no débito do sistema, medido como o número de transacções que são confirmadas por segundo. Nos cenários de baixa contenção, onde a taxa de cancelamentos é residual, os dois algoritmos possuem um desempenho bastante próximo, estando o débito do protocolo limitado pela eficiência do protocolo de confirmação. O algoritmo de ordem total necessita de uma fila adicional para garantir que todas as mensagens são entregues à aplicação de todas as réplicas pela mesma ordem. Esta ordem é confirmada pela última mensagem trocada pelo protocolo. A mensagem com os dados da transacção terá então de esperar que chegue a sua vez na fila para ser entregue à aplicação. Este tempo de espera poderá ser tanto maior quanto maior for o grau de replicação do sistema. Por oposição, na solução nativa do *Infinispan*, o terceiro passo de comunicação do protocolo leva à confirmação ou cancelamento imediato da transacção. Apesar deste custo adicional, em cenários de elevada contenção, o custo adicional de mensagens é compensado pelos ganhos que se obtêm por evitar cancelamentos por interbloqueio, pelo que mesmo com um grau de replicação 4 o nosso algoritmo ainda apresenta um débito superior. Quando o grau de replicação é apenas 2 (um valor realista), o débito do nosso algoritmo é cerca de 10 vezes superior. Para além disso, o nosso protocolo mantém um desempenho constante independentemente da carga do sistema, enquanto o 2PC se encontra fortemente dependente deste. É ainda importante solicitar que a solução baseada em difusão atómica apresenta resultados muito próximos do 2PC mesmo quando o cenário lhe é mais favorável, isto é, baixa contenção. Por outro lado, a Figura 3 apresenta o débito do sistema com a variação da taxa de operações de escrita. De um modo geral, o débito do sistema diminui com o aumento das operações de escritas para qualquer um dos algoritmos, pois é influenciado pela taxa de cancelamento e pela quantidade de dados que é necessária trocar entre as réplicas, que aumentam com o aumento do número de operações de escrita. No entanto, o nosso protocolo tem uma quebra menor com a variação da taxa de operações de escrita, obtendo-se vantagem sobre a solução



(a) Baixa contenção (b) Elevada contenção

Figura 2: Débito

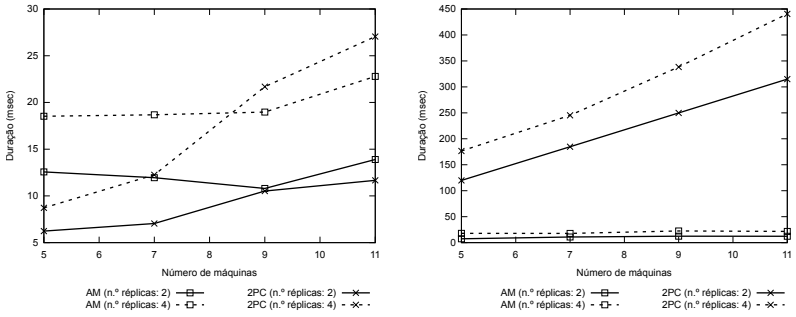


(a) Baixa contenção (b) Elevada contenção

Figura 3: Débito com variação na taxa média de operações de escrita

nativa. Esta vantagem chega a ser cerca de 700 vezes superior, para um cenário de elevada contenção, 100% de operações de escritas e 11 nós.

Latência Finalmente, a Figura 4 mostra a latência do processo de confirmação. Com baixa contenção, a duração da fase de conformação é limitada pela complexidade dos algoritmos na fase de entrega de mensagens. Daí o desempenho inferior do algoritmo baseado em difusão totalmente ordenada. Em cenários de elevada contenção, o algoritmo baseado em confirmação em duas fases é limitado pelo tempo necessário para detectar os interbloqueios, o que justifica a elevada latência desta solução. Como referido acima, o número de operações de escritas influencia a quantidade de dados que é necessário transferir. Desse modo, a variação da taxa de operações de escrita influencia a latência, que é superior à apresentada na Figura 4, chegando aos 100 milissegundos para o nosso protocolo e aos 10 segundos para o 2PC, no pior caso.



(a) Baixa contenção (b) Elevada contenção

Figura 4: Duração média da fase de confirmação

4.3 Discussão

Com base nos resultados anteriores, verifica-se que a utilização de difusão atómica acarreta vantagens em praticamente todos os cenários. Um aspecto limitativo da solução baseada em ordem total é o elevado custo do protocolo de ordenação no que se refere à entrega das mensagens, que torna a fila de entrega de mensagens ordenadas o principal ponto de estrangulamento do sistema. Julgamos que é possível mitigar parcialmente este problema através da optimização do código do algoritmo.

Além disso, eliminamos problemas complicados como o interboqueio do protocolo existente, sem sacrificar o desempenho do sistema, conseguindo-se um melhor desempenho em cenário onde os problemas do 2PC são mais evidentes.

De qualquer forma, os resultados mostram que usando uma solução baseada em difusão atómica conseguimos aumentar o desempenho do sistema adicionando novos nós, o que não acontece usando o 2PC.

5 Trabalho Relacionado

Este trabalho resulta da confluência de diferentes linhas de investigação com múltiplos pontos de contacto entre si: o desenvolvimento de sistemas de cache em memória distribuída, o desenvolvimentos de sistemas de replicação de bases de dados, e os sistemas de memória transaccional distribuída.

Os sistemas de cache em memória distribuída emergiram como ferramentas para aumentar o desempenho de sistemas que necessitam de aceder a grandes quantidades de informação com baixa latência. Os primeiros sistemas não consideravam suporte para transacções [17,11], mas sistemas mais recentes têm vindo a incorporar este suporte. O sistema *Sinfonia* [5] suporta transacções e replicação, mas pressupõe que as transacções são estáticas (o seu conjunto de leitura

e escrita é conhecido á partida). O sistema *TxCache* [20] assegura que os utilizadores da cache observam um corte coerente, mas não aborda o problema da replicação parcial.

A área da replicação de bases de dados é rica em algoritmos para manter as várias réplicas coerentes em ambientes transaccionais. Muitos destes sistemas usam replicação total [19,15], mas existem também vários sistemas que suportam replicação parcial [25,22,24]. Destes sistemas, destacamos o *P-Store* [23], que propõe algoritmos genuínos, isto é, em que apenas as réplicas envolvidas numa dada transacção participam na coordenação necessária para a sua confirmação. O algoritmo do *P-Store* é no entanto mais complexo e oneroso que o nosso, pois oferece garantias de coerências mais fortes. A nossa solução concretiza um conjunto de optimizações adaptadas ao modelo de "*Read Committed*" adoptado no *Infinispan*.

Finalmente os sistemas de memória transaccional distribuída emergem como uma extensão dos sistemas de memória transaccional por software desenvolvidos para máquinas multi-processador [8,13,9]. A grande maioria destes sistemas, ou não considera tolerância a faltas [16,7], ou considera apenas replicação total [10].

6 Conclusão

Neste artigo apresentámos uma solução para o suporte à replicação parcial em sistemas de cache em memória transaccional distribuída. A solução proposta é inspirada em soluções desenvolvidas no contexto da replicação de bases de dados, que foram adaptadas para suportar modelos de coerência mais fracos. A solução resultante suporta replicação parcial genuína (isto é, apenas as réplicas dos dados actualizados numa determinada transacção participam na sua confirmação), o que permite distribuir a carga no sistema. A solução foi concretizada no sistema *Infinispan* e o seu desempenho comparado com o suporte nativo oferecido pela plataforma. Ao contrário da solução nativa, baseada no protocolo de confirmação em duas fases, a nossa solução evita o interbloqueio. A avaliação de desempenho mostra que a solução aqui proposta permite obter ganhos várias ordens superiores. Como trabalho futuro pretendemos estender o nosso trabalho para suportar outros modelos de coerência, tal como "*Repeatable Read*" e "*Serializability*".

Agradecimentos

Os autores do artigo agradecem aos elementos do Grupo de Sistemas Distribuídos do INESC-ID Diego Didona e Sebastiano Peluso pelo apoio prestado durante a preparação deste trabalho.

Referências

1. Ehcache, http://ehcache.org/documentation/overview.html
2. Infinispan, http://www.jboss.org/infinispan

3. Jgroups, http://www.jgroups.org
4. Oracle Coherence, http://coherence.oracle.com/display/COH/Oracle+Coherence+Knowledge+Base+Home
5. Aguilera, M.K., Merchant, A., Shah, M., Veitch, A., Karamanolis, C.: Sinfonia: a new paradigm for building scalable distributed systems. SIGOPS '07 41, 159–174
6. Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., O'Neil, P.: A critique of ansi sql isolation levels. SIGMOD '95 24, 1–10 (May 1995)
7. Bocchino, R.L., Adve, V.S., Chamberlain, B.L.: Software transactional memory for large scale clusters. In: PPoPP '08. pp. 247–258. ACM, New York, NY, USA
8. Cachopo, J.a., Rito-Silva, A.: Versioned boxes as the basis for memory transactions. Sci. Comput. Program. 63, 172–185 (December 2006)
9. Carvalho, N., Romano, P., Rodrigues, L.: Asynchronous Lease-Based Replication of Software Transactional Memory. In: ACM/IFIP/USENIX 11th Middleware Conference. Bangalore, India (Nov 2010)
10. Couceiro, M., Romano, P., Carvalho, N., Rodrigues, L.: D2stm: Dependable distributed software transactional memory. In: PRDC '09. pp. 307–313. IEEE Computer Society, Washington, DC, USA (2009)
11. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. SIGOPS '07 41, 205–220 (October 2007)
12. Defago, X., Schiper, A., Urban, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. 36(4), 372–421 (2004)
13. Dice, D., Shalev, O., Shavit, N.: Transactional Locking II. In: In Proc. of the 20th Intl. Symp. on Distributed Computing (2006)
14. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edn. (1992)
15. Kemme, B., Alonso, G.: A suite of database replication protocols based on group communication primitives. In: ICDCS '98. p. 156. IEEE Computer Society (1998)
16. Kotselidis, C., Ansari, M., Jarvis, K., Luján, M., Kirkham, C., Watson, I.: DiSTM: A software transactional memory framework for clusters. In: ICPP '08. pp. 51–58
17. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. SIGOPS '10 44, 35–40 (April 2010)
18. Lamport, L.: Ti clocks, and the ordering of events in a distributed system. Commun. ACM 21, 558–565 (July 1978)
19. Pedone, F., Guerraoui, R., Schiper, A.: The database state machine approach. Distributed and Parallel Databases 14(1), 71–98 (2003)
20. Ports, D.R.K., Clements, A.T., Zhang, I., Madden, S., Liskov, B.: Transactional consistency and automatic management in an application data cache. In: OSDI'10. pp. 1–15. USENIX Association, Berkeley, CA, USA (2010)
21. Schiper, A., Birman, K., Stephenson, P.: Lightweight causal and atomic group multicast. ACM Trans. Comput. Syst. 9, 272–314 (August 1991)
22. Schiper, N., Schmidt, R., Pedone, F.: Optimistic Algorithms for Partial Database Replication. In: OPODIS '06). pp. 81–93 (2006)
23. Schiper, N., Sutra, P., Pedone, F.: P-store: Genuine partial replication in wide area networks. In: SRDS '10. pp. 214–224. IEEE Computer Society, Washington, DC, USA (2010)
24. Serrano, D., Patino-Martinez, M., Jimenez-Peris, R., Kemme, B.: Boosting Database Replication Scalability through Partial Replication and 1-Copy-Snapshot-Isolation. In: PRDC '07. pp. 290–297. Washington, DC, USA (2007)
25. Sousa, A., Pedone, F., Moura, F., Oliveira, R.: Partial Replication in the Database State Machine. In: NCA '01. pp. 298–309. IEEE CS (October 2001)