D^{2}STM: Memória Transacional em Software Distribuída e Confiável*

Maria Couceiro, Paolo Romano, Nuno Carvalho, and Luis Rodrigues

INESC-ID/IST

maria.couceiro@ist.utl.pt, romanop@gsd.inesc-id.pt,
nonius@gsd.inesc-id.pt, ler@ist.utl.pt

Resumo Os sistemas de Memória Transacional em Software (MTS) têm vindo a ganhar relevo no desenvolvimento de aplicações concorrentes. No entanto, o recurso à distribuição e à replicação neste contexto não se encontra ainda devidamente explorado. Este artigo apresenta o D^{2}STM, um sistema de MTS replicada que rentabiliza os recursos disponíveis nos vários nós de um sistema distribuído. A coerência da MTS replicada é assegurada de modo transparente, mesmo na presença de faltas no sistema. No D^{2}STM, as transacções são processadas de forma autónoma por um único nó, evitando-se qualquer comunicação entre réplicas durante a sua execução. Um processo distribuído de certificação não-bloqueante, ao qual se deu o nome de BFC (*Bloom Filter Certification*), assegura a coerência no momento de *confirmação* da transacção. O BFC explora os mecanismos de codificação dos filtros de Bloom para reduzir substancialmente o custo inerente à coordenação das várias réplicas (à custa de um aumento da probabilidade de uma transacção ser abortada, com um valor que pode ser ajustado pelo utilizador). Recorrendo a uma bancada experimental mostra-se que o BFC permite atingir ganhos consideráveis.

Abstract. Software Transactional Memory (STM) systems have emerged as a powerful paradigm to develop concurrent applications. This paper addresses the problem of building a distributed and replicated STM system. We present D^{2}STM, a replicated STM, where consistency is ensured in a transparent manner, even in the presence of failures. In D^{2}STM transactions are autonomously processed on a single node, avoiding any replica inter-communication during transaction execution. Consistency is enforced at transaction commit time by a non-blocking distributed certification scheme, which we name BFC (Bloom Filter Certification). BFC exploits a novel Bloom Filter-based encoding mechanism that permits to significantly reduce the overhead of replica coordination (at the cost of a user tunable increase in the probability of transaction abort). Experimental results based on STM benchmarks show that the BFC scheme permits to achieve remarkable performance gains.

* Este trabalho foi parcialmente suportado pelo projecto Pastramy (PTDC/EIA/72405/2006).

1 Introdução

Os sistemas de Memória Transacional em Software (MTS) têm vindo a ganhar relevância no desenvolvimento de aplicações concorrentes [1,2]. Os programadores que utilizam MTSs não necessitam de lidar explicitamente com mecanismos de controlo de concorrência, tendo apenas de identificar a sequência de instruções, ou transacções, que acedem ou modificam objectos partilhados de forma concorrente e que necessitam de ser executadas de forma atómica. Aumenta-se, assim, a fiabilidade do código, ao mesmo tempo que se diminui o tempo de desenvolvimento.

Apesar do interesse que as MTSs têm suscitado, só muito recentemente se começou a abordar o problema da construção de arquitecturas distribuídas para este tipo de sistemas [3,4,5]. Para além disso, as soluções anteriormente propostas recorrem à replicação como forma de melhorar o desempenho e não exploram o facto de esta poder também ser usada para obter tolerância a faltas. No entanto, este é um aspecto central no desenho das MTS distribuídas, uma vez que a probabilidade de ocorrência de falhas aumenta com o número de nós, sendo incontornável num sistema de grande dimensão. As garantias de coerência forte e tolerância a faltas são também essenciais quando as MTSs são usadas para aumentar a robustez das aplicações orientadas a serviços. Como exemplo, cita-se o sistema FenixEDU [6], uma aplicação web na qual a semântica transacional das operações é suportada por uma MTS.

Neste trabalho apresenta-se o D^{2}STM, um sistema de MTS tolerante a faltas e distribuído que permite tirar partido dos recursos computacionais disponíveis num *cluster*, usando a interface convencional de uma MTS e assegurando de modo transparente a coerência dos dados de forma não bloqueante, mesmo em caso de falhas.

O esquema de sincronização de réplicas usado no D^{2}STM é inspirado por trabalhos recentes na área da replicação de bases de dados [7,8,9], os quais recorrem às propriedades da primitiva de difusão atómica [10] para manter a coerência entre réplicas através de um processo de certificação distribuído. Este tipo de certificação permite que as transacções sejam executadas localmente de forma optimista, sendo a coerência das réplicas (normalmente *1-Copy serializability*) assegurada no momento da confirmação da transacção. Esta fase de certificação distribuída recorre às propriedades da primitiva de difusão atómica para que as transacções tenham uma ordem de seriação comum a todas as réplicas. Para além disso, apenas uma réplica executa toda a transacção (de escrita), enquanto que as restantes têm somente de a validar e guardar as actualizações resultantes. Esta abordagem permite uma capacidade de escala elevada, mesmo na presença de padrões de utilização dominados por operações de escrita, desde que a taxa de conflitos entre transacções não atinja valores muito elevados [7].

Apesar do uso deste tipo de certificação parecer o mais indicado para aplicar em MTSs, resultados anteriores [11] (confirmados pelos resultados experimentais apresentados neste trabalho) mostram que o uso de primitivas de difusão atómica pode ser extremamente penalizante neste contexto. Uma vez que, num sistema baseado em MTSs, não existem necessariamente acessos ao disco, nem

o processamento relacionados com a análise de comandos SQL ou com a optimização de planos de execução, o tempo de execução de uma transacção numa MTS é significativamente mais curto do que num sistema de base de dados clássico. Com o objectivo de minimizar o impacto da coordenação inter-réplicas, o D^{2}STM introduz um novo processo de certificação, com o nome de BFC (*Bloom Filter Certification*). O BFC tira partido das técnicas de codificação dos filtros de Bloom para reduzir de forma significativa quer o processamento associado à certificação distribuída, quer o tamanho das mensagens trocadas através da primitiva de difusão atómica. Estas melhorias são atingidas com o sacrifício de um aumento marginal (mas configurável pelo utilizador) da probabilidade de uma transacção abortar.

O D^{2}STM foi construído sobre a JVSTM [12], uma biblioteca MTS que suporta controlo de concorrência multi-versão e, como consequência, oferece um excelente desempenho para transacções em que só se executam leituras, uma vez que estas se encontram protegidas contra a possibilidade de abortarem devido a conflitos (locais ou remotos). Através de uma extensa avaliação experimental, baseada não só em testes localizados mas também em bancadas de teste mais complexas, mostra-se que o D^{2}STM permite atingir ganhos consideráveis no desempenho, sem aumentar de forma significativa a probabilidade de uma transacção abortar.

O resto do artigo está organizado da seguinte forma. Na Secção 2 discute-se o trabalho relacionado. Na Secção 3 apresenta-se o modelo do sistema e na Secção 4 mostra-se uma visão geral da arquitectura do sistema, bem como a integração da JVSTM no D^{2}STM. O BFC é apresentado na Secção 5 e na Secção 6 mostram-se os resultados da avaliação experimental. Finalmente, a Secção 7 conclui o artigo.

2 Trabalho Relacionado

Nesta secção faz-se uma síntese do trabalho relevante para a construção do D^{2}STM. Começa-se por analisar sistemas de MTS distribuída, tendo em conta alguns aspectos fundamentais, como a tolerância a faltas e desempenho. Apresenta-se também uma discussão das vantagens e desvantagens de recentes algoritmos de replicação de bases de dados quando aplicados no contexto de MTSs distribuídas.

2.1 MTSs Distribuídas

As únicas soluções de MTSs distribuídas das quais temos conhecimento estão descritas em [3,4,5] e, como já foi mencionado na introdução, nenhuma delas tira partido da replicação como meio de assegurar coerência e disponibilidade quando ocorrem faltas. Por oposição, a tolerância a faltas foi tida em conta durante o desenho do D^{2}STM, estando estes mecanismos (como, por exemplo, a difusão atómica) integrados com um novo e eficiente processo distribuído de certificação de transacções. Segue-se uma descrição das principais diferenças entre os processos de manutenção de coerência entre réplicas adoptados por estas soluções.

O trabalho apresentado em [3] utiliza um algoritmo multi-versão distribuído (DMV, *Distributed Multi Versioning*) e explora a presença simultânea de várias versões do mesmo conjunto de dados nas diferentes réplicas. O DMV permite que transacções de leitura sejam executadas em paralelo com as de escrita (podendo ocorrer conflitos), à semelhança de outros algoritmos de controlo de concorrência multi-versão centralizados [13] (onde se inclui a JVSTM [12]). No entanto, cada réplica mantém uma única versão dos dados e atrasa a sua actualização para diminuir a probabilidade de abortar transacções de leitura em execução. O D^{2}STM tem como base uma MTS multi-versão (JVSTM) que, para cada objecto transacional, mantém um número de versões suficiente para garantir que nenhuma transacção de leitura é abortada. Adicionalmente, o DMV requer que cada transacção que pretende confirmar adquira previamente um testemunho (*token*) único no sistema para obter uma seriação global das transacções que confirmam, introduzindo um atraso considerável [4]. Por oposição, no D^{2}STM as réplicas têm apenas de executar uma validação local da transacção após a fase de coordenação, que tem como base uma primitiva de difusão atómica e pode ser executada de forma concorrente por todas as réplicas.

O trabalho descrito em [4] não recorre a algoritmos multi-versão mas, analogamente a [3], utiliza um mecanismo distribuído de exclusão mútua. Estes mecanismos têm como objectivo assegurar que duas réplicas nunca irão confirmar simultaneamente transacções com conflitos entre si. O uso de múltiplas *leases* com base no conjunto de dados acedidos pelas transacções permite atenuar os problemas de desempenho causados pela seriação da fase de confirmação distribuída. No entanto, a sua atribuição é coordenada por uma única réplica, o que diminui a capacidade de escala da solução (comprovado pelo facto de a avaliação experimental em [4] não apresentar resultados para mais de quatro réplicas).

Finalmente, o Cluster-STM, apresentado em [5], centra-se no problema de particionar o conjunto de dados pelos nós de um MTS distribuído de grande escala. Cada item de dados transaccional tem atribuído um nó responsável não só por manter a cópia principal dos dados (isto é, aquela que corresponde sempre à versão mais actual dos dados), mas também por sincronizar os acessos de transacções remotas que possam gerar conflitos. No entanto, os esquemas de replicação são delegados para o nível da aplicação, aumentando assim a sua complexidade. Para além disso, processadores no mesmo nó ou em nós diferentes são tratados da mesma forma, não sendo explorada a partilha de memória entre múltiplos processadores para tornar mais rápida a comunicação dentro do mesmo nó.

2.2 Replicação em Bases de Dados

A maioria dos algoritmos de replicação de bases de dados recentes [7,8,9] recorrem a uma primitiva de difusão atómica [10,14], tipicamente disponibilizada por um Serviço de Comunicação em Grupo (SCG) [15,16]. Esta primitiva assegura uma ordem global de seriação de transacções, sem os problemas de interbloqueio e falta de capacidade de escala que caracterizam os mecanismos clássicos de replicação [17].

As abordagens optimistas, como por exemplo [7], recorrem à execução das transacções numa única réplica e à sua posterior validação através de um processo de certificação global (baseado na difusão atómica) que permite detectar conflitos entre transacções concorrentes. Estas abordagens podem ser divididas em dois grandes grupos [8], nomeadamente (*i*) nas que não incluem um processo de votação (envia-se tanto o conjunto de escritas como o de leituras de cada transacção) e (*ii*) nas que o incluem (onde só se envia o conjunto de escritas da transacção mas é necessário transmitir uma segunda mensagem durante a fase de confirmação [14]). Dado que o tempo de execução de uma transacção é, em média, mais curto numa MTS do que num sistema de base de dados, o atraso induzido pela coordenação de réplicas é relativamente elevado [11]. Isto implica que os processos de certificação que incluem votação não sejam os mais indicados para aplicar neste contexto, uma vez que introduzem um passo extra na comunicação. Por outro lado, iremos mostrar na avaliação experimental que a eficiência dos protocolos sem votação é profundamente afectada pelo tamanho do conjunto de leituras das transacções.

O esquema de coordenação de réplicas utilizado pelo D^{2}STM, ao qual se deu o nome de BFC (*Bloom Filter Certification*), pode ser classificado como um algoritmo de certificação sem votação. O conjunto de leituras das transacções é comprimido num filtro de Bloom de modo a ser possível fazer apenas uma difusão atómica e não congestionar a rede com mensagens demasiado extensas. Isto é conseguido à custa de um ligeiro aumento na probabilidade de uma transacção ser abortada (definido pelo utilizador).

3 Modelo do Sistema

Consideramos um sistema distribuído assíncrono clássico com $\Pi = \{p_1, \ldots, p_n\}$ processos que comunicam através de mensagens e podem falhar de acordo com o modelo de falha por paragem. Assumimos que a maioria dos processos está correcta e que o sistema assegura um nível de sincronismo suficiente (por exemplo, através de um detector de falhas $\diamondsuit S$) que permite concretizar um serviço de difusão atómica com as propriedades de validade, acordo, integridade e ordem total uniforme [10].

O critério de consistência para o estado das instâncias MTSs replicadas assegurado pela D^{2}STM corresponde a *1-copy serializability* de leituras e escritas de dados transaccionais [13], ou seja, o histórico da execução de um conjunto de transacções num conjunto de réplicas é equivalente ao histórico da sua execução em série numa única instância de MTS não replicada.

4 Arquitectura do D^{2}STM

4.1 Componentes

Como a Figura 1 ilustra, um nó do sistema D^{2}STM encontra-se estruturado em quatro camadas lógicas. A camada de baixo é materializada por um serviço de

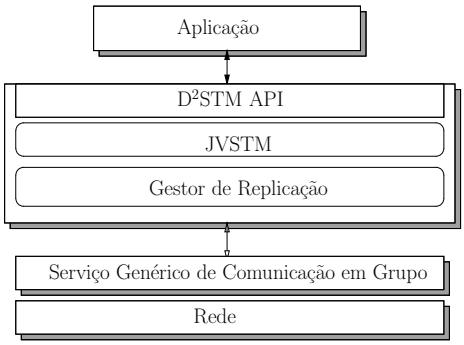


Figura 1. Componentes de uma réplica D^{2}STM.

comunicação de grupo (SCG) [10] que fornece uma primitiva de difusão atómica, entre outras. A nossa concretização usa um SCG genérico [18], que suporta várias concretizações do SCG (todos os testes descritos neste artigo foram realizados com o SCG Appia [15]). O componente principal do D^{2}STM é representado pelo Gestor de Replicação que concretiza o protocolo de coordenação distribuído necessário para assegurar a coerência das réplicas (isto é, *1-copy serializability*); na Secção 5 encontra-se uma descrição detalhada deste componente. Por fim, a camada superior da D^{2}STM é um contentor que intercepta chamadas a marcadores de transacção (isto é, indicações para iniciar e terminar transacções) do nível da aplicação sem interferir com os acessos da aplicação (leitura e escrita) aos objectos que são geridos directamente pela camada inferior (JVSTM). Esta abordagem permite à D^{2}STM estender o modelo clássico de programação de MTSs de forma transparente, não sendo necessário realizar modificações significativas às aplicações JVSTM pré-existentes.

4.2 Integração com a JVSTM

A JVSTM usa um algoritmo multi-versão que se baseia na abstracção de *versioned box* (VBox) para guardar o estado mutável de um programa concorrente. Uma VBox é um contentor que guarda uma sequência de valores etiquetados, isto é, o seu histórico. Cada um destes valores corresponde a uma alteração à *box* feita por uma transacção confirmada e é etiquetada com o número de versão (inteiro incrementado a cada confirmação) da transacção correspondente. Cada transacção é iniciada com o número de versão da última transacção confirmada com sucesso (*commitTimestamp*). Esta informação é usada durante a execução da transacção (para identificar os valores a ler das VBoxes) e também quando esta se prepara para confirmar, durante a fase de validação (para determinar o conjunto de transacções concorrentes e verificar se ocorreu algum conflito).

A abordagem optimista da JVSTM consiste em diferir as actualizações de uma transacção para fazer a detecção de conflitos apenas durante a confirmação, verificando se algumas das *VBoxes* lidas por uma transacção T foram modificadas por outra transacção T' com um número de versão superior. Em caso

afirmativo, T é abortada. Caso contrário, o número de versão de T é actualizado com o valor de *commitTimestamp* e os novos valores das VBoxes que actualizou são atomicamente guardados nas respectivas VBoxes.

De modo a aumentar o desempenho, o protocolo de coordenação de réplicas do D^{2}STM, ou seja, o BFC, está intrinsecamente integrado no mecanismo de gestão de versões da JVSTM. Os pormenores desta integração podem ser consultados em [19].

5 Certificação com Filtros de Bloom

A certificação com filtros de Bloom (BFC) é um novo processo de certificação sem votação que explora as propriedades destes filtros [20], de modo a reduzir drasticamente o custo da fase de certificação distribuída com o sacrifício de um aumento reduzido (e controlado) no risco de uma transacção abortar.

Um filtro de Bloom que representa um conjunto $S = \{x_1, x_2, \ldots, x_n\}$ com n elementos consiste num vector de m dígitos binários inicializados a 0 [21] . O filtro usa k funções de dispersão $h_1, \ldots, h_k$ no intervalo $\{1, \ldots, m\}$. Se o conjunto não estiver vazio, para cada elemento $x \in S$, os dígitos $h_i(x)$ são colocados a 1 para $1 \leq i \leq k$. Para verificar se um item y pertence a S, todos as posições $h_i(y)$ têm de estar a 1. Se isto não se verificar, então y não pertence a S. Caso contrário, assume-se que y pertence a S, com uma probabilidade de erro conhecida. A taxa de falsos positivos para uma única interrogação a um filtro depende do número de dígitos usados por item m/n e do número de funções de dispersão k, de acordo com a seguinte equação:

$$f = (1 - e^{-kn/m})^k \tag{1}$$

onde o número de funções de dispersão k óptimo que minimiza a probabilidade de falsos positivos dado por m e n é igual a:

$$k = \lceil ln\ 2 \cdot m/n \rceil \tag{2}$$

No BFC, as transacções de leitura são executadas localmente e a confirmação é feita sem nenhum custo adicional, tirando-se partido do algoritmo multi-versão da JVSTM para assegurar que estas transacções nunca são abortadas (devido a conflitos locais ou remotos).

Após a sua execução, uma transacção com um conjunto de escritas não nulo (ou seja, que actualizou pelo menos uma VBox) passa por uma fase de validação local (de modo a evitar a fase distribuída do processo de certificação em transacções quando é possível determinar localmente o resultado). Se a transacção passa esta fase, o Gestor de Replicação codifica o conjunto de leituras da transacção (ou seja, o conjunto de identificadores das *VBoxes* lidas durante a sua execução) num filtro de Bloom e, juntamente com o conjunto de escritas da transacção (este não codificado no filtro), envia-o através de difusão atómica.

Tal como nos protocolos de certificação sem votação clássicos, as transacções de escrita são validadas aquando da sua entrega (por difusão atómica). Verifica-se, então, se o filtro de Bloom de uma transacção T_x contém algum item ac-

tualizado por transacções com um número de versão maior que T_x. Se isto se verificar, então T_x é abortada. Caso contrário, T_x pode ser confirmada.

Uma vez que a fase de validação de uma transacção T_x necessita dos conjuntos de escritas das transacções concorrentes já confirmadas, o Gestor de Replicação guarda os identificadores das *VBoxes* actualizadas por elas. De modo a evitar manter em memória informação sobre transacções que já não afectam nenhuma transacção activa, recorremos a um processo de reciclagem de memória distribuído (análogo ao usado em [22]) no qual cada réplica dá a conhecer o menor número de versão das suas transacções activas (juntando-o às mensagens de validação das transacções), sendo assim possível qualquer réplica determinar qual a transacção mais antiga em execução no sistema.

Para uma transacção T_x ser abortada devido a um falso positivo, basta que este ocorra em qualquer um dos itens actualizados por transacções concorrentes. Logo, para determinar o tamanho do filtro de uma transacção seria necessário saber exactamente o número de interrogações q que lhe iriam ser feitas durante a fase de validação, de modo a que nunca se exceda a taxa de aborto máxima definida pelo utilizador ($maxAbortRate$). Por outro lado, no momento em que T_x inicia a fase de validação, é impossível prever o número e o tamanho dos conjuntos de escritas de transacções que entretanto confirmaram. No entanto, erros que possam ocorrer na estimativa de q resultam apenas em desvios (positivos ou negativos) na $maxAbortRate$. Por isso, q é estimado através da média do número de interrogações feitas a um filtro durante a fase de validação das últimas t transacções (em que t é um valor pré-definido), já que este número é facilmente obtido por todas as réplicas. A partir de q, podemos determinar o número de dígitos m, considerando que os falsos positivos para cada interrogação distinta são eventos independentes e identicamente distribuídos que geram um processo de Bernoulli [23]. Assim, a probabilidade de uma transacção abortar devido a um falso positivo no processo de validação baseado em filtros de Bloom pode ser representada por:

$$maxAbortRate = 1 - (1 - f)^q$$

que, combinada com as equações 1 e 2, nos permite estimar m como:

$$m = \left\lceil -n \, \frac{log_2(1 - (1 - maxAbortRate)^{\frac{1}{q}})}{ln\ 2} \right\rceil$$

Podemos provar (informalmente) a correcção do BFC tendo em consideração que (i) as réplicas validam todas as transacções de escrita pela mesma ordem (determinada pela primitiva de difusão atómica) e que (ii) o processo de validação é determinista, uma vez que todas elas utilizam o mesmo conjunto de funções de dispersão na construção/descodificação dos filtros de Bloom. Assim, a ocorrência de falsos positivos durante o processo de validação apenas resulta no aumento da probabilidade de uma transacção abortar e nunca em incoerências no estado das réplicas.

6 Avaliação

Apresentamos agora os resultados da avaliação experimental realizada. Esta pretende determinar os ganhos no desempenho quando o BFC é aplicado num sistema de MTS distribuído real, ou seja, usando um protótipo do D^{2}STM com cargas sintéticas e com padrões de carga complexos. Os testes foram realizados num sistema com 8 nós, cada um deles equipado com um Intel QuadCore Q6600 a 2.40GHz com 8 GB de RAM e Linux 2.6.27.7 ligado via Ethernet Gigabit privada. A concretização do protocolo de difusão atómica usada foi baseada no algoritmo clássico que usa um sequenciador [14,10].

Começa-se por considerar uma carga sintética (adaptada do *Bank Benchmark* originalmente usada para avaliar a DSTM2 [24]). Cada réplica é iniciada com um vector de $numThreads \cdot numMachines \cdot 10.000$ itens e cada fio de execução acede a um fragmento distinto de 10.000 elementos do vector, lendo-os a todos e aleatoriamente actualizando um número de elementos uniformemente distribuído no intervalo [50, 100]. Como diferentes fios de execução acedem a fragmentos do vector que não se sobrepõem, os falsos positivos da validação com filtros de Bloom são a única razão pela qual as transacções são abortadas. O gráfico da Figura 2 mostra a percentagem de transacções abortadas quando se usa o BFC com *maxAbortRate* de 1%, 5%, 10%, variando-se o número de réplicas entre 1 e 8 (com 4 fios de execução em cada réplica), e ilustra a correspondência entre a previsão analítica e os resultados experimentais.

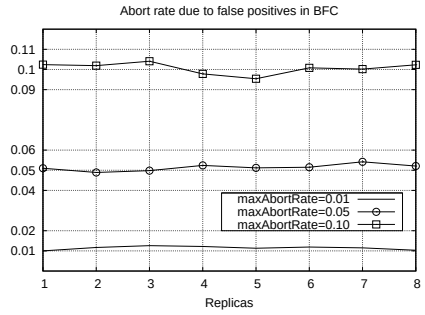
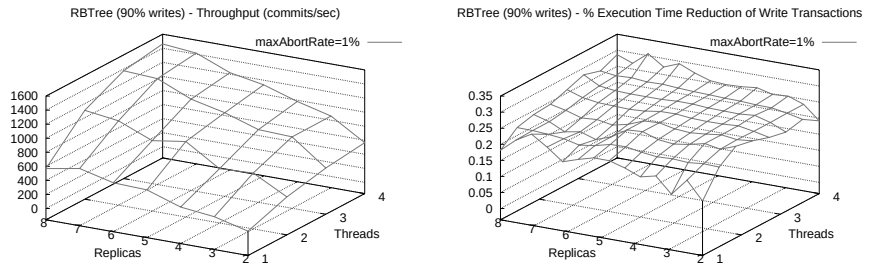


Figura 2. Taxa de cancelamento de transacções devido a falsos positivos.

Apresentam-se agora resultados para uma *micro-bancada* de teste mais complexa, denominada *Red Black Tree* (novamente adaptada de uma implementação usada para avaliar a DSTM2 [24]). Existem três tipos diferentes de transacções: i) leitura que executa uma sequência de pesquisas, ii) escrita que executa uma sequência de pesquisas e inserções, e iii) escrita que executa uma sequência de pesquisas e remoções. A bancada foi configurada (os detalhes de configuração usados nestes e noutros testes podem ser consultados em [19]) de modo a que a probabilidade de contenção de leitura-escrita seja baixa e com um padrão de

utilização de 90% de transacções de escrita. A Figura 3(a) mostra o débito do sistema variando o número de réplicas entre 2 e 8 e o número de fios de execução entre 1 e 4. É possível observar uma melhoria linear no desempenho com o aumento do número de réplicas, chegando mesmo a duplicar entre a utilização de 2 e 6 réplicas. A Figura 3(b) mostra a redução do tempo de execução de transacções de escrita com o BFC quando comparado com um processo de certificação sem votação clássico, em tudo semelhante ao BFC, mas no qual se enviam os conjuntos de leitura e escrita sem qualquer tipo de compressão. Pode-se observar que os ganhos obtidos com o BFC variam entre os 20% e os 30%.

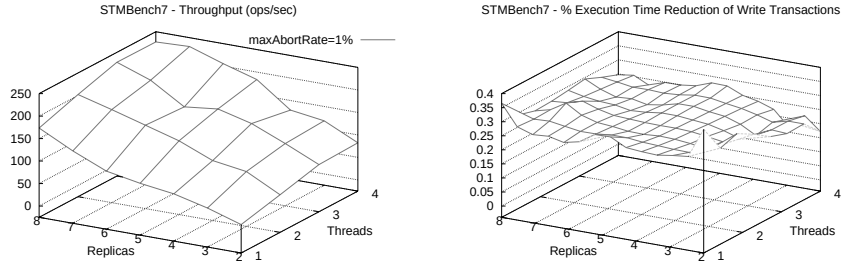


(a) Débito (b) Redução no tempo de execução (%)

Figura 3. Red Black Tree, 90% de escritas, $maxAbortRate$=1%

Apresentam-se finalmente resultados obtidos usando a STMBench7 [25]. A Figura 4 mostra o desempenho do sistema quando usada a carga denominada *"read dominated with long traversals"*. O gráfico mostra o débito para um número de réplicas entre 2 e 8, cada uma com 1 a 4 fios de execução. Os resultados observados no aumento do débito são coerentes com os obtidos na bancada de testes anterior. A Figura 4(b) mostra os ganhos no desempenho obtidos com o BFC quando comparado com o processo de certificação sem votação clássico já referido. É possível observar que o tempo de execução das transacções de escrita (ou seja, as que passam pelo processo de certificação distribuído) apresenta reduções que variam entre 20% e 40%. Um facto interessante sublinhado por estes testes é o facto de o BFC atingir ganhos no desempenho consideráveis mesmo com um aumento na probabilidade de uma transacção abortar muito pequeno (1%).

Podemos assim concluir que o BFC concretizado na D^{2}STM assegura tolerância a faltas, torna possível o uso de réplicas adicionais para melhorar o débito do sistema (principalmente na presença de cargas em que predominem operações de leitura) e, por fim, permite o uso de um processo de certificação sem votação mais rápido na presença de cargas com conjuntos de leitura de grandes dimensões.



(a) Débito (b) Redução no tempo de execução (%)

Figura 4. STMBench7, *read dominated with long traversals*, *maxAbortRate*=1%

7 Conclusões

Este trabalho apresentou o D^{2}STM, um sistema de memória transaccional em software distribuída que assegura coerência forte e alta disponibilidade mesmo na presença de falhas em (uma minoria de) réplicas. O BFC, algoritmo que mantém a coerência entre as réplicas utilizado pelo D^{2}STM, tira partido dos filtros de Bloom para reduzir de modo significativo o custo associado à fase de certificação das transacções. Para além disso, graças à integração com uma MTS multi-versão, o D^{2}STM processa as transacções de leitura localmente sem o risco destas serem abortadas devido a algum conflito local ou remoto, evitando-se ao mesmo tempo a comunicação desnecessária entre réplicas.

Referências

1. Herlihy, M., Luchangco, V., Moir, M., Scherer, III, W.N.: Software transactional memory for dynamic-sized data structures. In: Proc. of the Symposium on Principles of Distributed Computing (PODC), ACM (2003) 92–101
2. Fraser, K.: Practical lock freedom. PhD thesis, Cambridge University Computer Laboratory (2003) Also available as Technical Report UCAM-CL-TR-579.
3. Manassiev, K., Mihailescu, M., Amza, C.: Exploiting distributed version concurrency in a transactional memory cluster. In: Proc. of the Symposium on Principles and Practice of Parallel Programming (PPOPP), ACM (2006) 198–208
4. Kotselidis, C., Ansari, M., Jarvis, K., Lujan, M., Kirkham, C., Watson, I.: DiSTM: A software transactional memory framework for clusters. In: Proc. of the International Conference on Parallel Processing (ICPP). (2008) 51–58
5. Bocchino, R.L., Adve, V.S., Chamberlain, B.L.: Software transactional memory for large scale clusters. In: Proc. of the Symposium on Principles and Practice of Parallel Programming (PPOPP), ACM (2008) 247–258
6. Carvalho, N., Cachopo, J., Rodrigues, L., Rito Silva, A.: Versioned transactional shared memory for the FenixEDU web application. In: Proc. of the Workshop on Dependable Distributed Data Management (WDDDM), ACM (2008)

7. Pedone, F., Guerraoui, R., Schiper, A.: The database state machine approach. Distributed and Parallel Databases **14**(1) (2003) 71–98
8. Kemme, B., Alonso, G.: A suite of database replication protocols based on group communication primitives. In: Proc. of the International Conference on Distributed Computing Systems (ICDCS), IEEE Computer Society (1998) 156
9. Patino-Martínez, M., Jiménez-Peris, R., Kemme, B., Alonso, G.: Scalable replication in database clusters. In: Proc. of the International Conference on Distributed Computing (DISC), Springer-Verlag (2000) 315–329
10. Defago, X., Schiper, A., Urban, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Computing Surveys **36**(4) (2004) 372–421
11. Romano, P., Carvalho, N., Rodrigues, L.: Towards distributed software transactional memory systems. In: Proc. of the Workshop on Large-Scale Distributed Systems and Middleware (LADIS). (2008)
12. Cachopo, J., Rito-Silva, A.: Versioned boxes as the basis for memory transactions. Sci. Comput. Program. **63**(2) (2006) 172–185
13. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison-Wesley (1987)
14. Guerraoui, R., Rodrigues, L.: Introduction to Reliable Distributed Programming. Springer (2006)
15. Miranda, H., Pinto, A., Rodrigues, L.: Appia, a flexible protocol kernel supporting multiple coordinated channels. In: Proc. International Conference on Distributed Computing Systems (ICDCS), IEEE (2001) 707–710
16. Amir, Y., Danilov, C., Stanton, J.: A low latency, loss tolerant architecture and protocol for wide area group communication. In: Proc. of the International Conference on Dependable Systems and Networks (DSN). (2000)
17. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: Proc. of the Conference on the Management of Data (SIGMOD), ACM (1996) 173–182
18. Carvalho, N., Pereira, J., Rodrigues, L.: Towards a generic group communication service. In: Proc. of the International Symposium on Distributed Objects and Applications (DOA). (2006)
19. Couceiro, M., Romano, P., Carvalho, N., Rodrigues, L.: D^2STM: Dependable distributed software transactional memory. Technical Report 30/2009, INESC-ID (2009)
20. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7) (1970) 422–426
21. Broder, A., Mitzenmacher, M.: Network Applications of Bloom Filters: A Survey. Internet Mathematics **1**(4) (2003) 485–509
22. Perez-Sorrosal, F., Patino-Martinez, M., Jimenez-Peris, R., Kemme, B.: Consistent and scalable cache replication for multi-tier J2EE applications. In: Proc. of the International Conference on Middleware (Middleware), Springer-Verlag (2007) 328–347
23. Bertsekas, D.P., Tsitsiklis, J.N.: Introduction to Probability. Athena Scientific (2002)
24. Herlihy, M., Luchangco, V., Moir, M.: A flexible framework for implementing software transactional memory. SIGPLAN Not. **41**(10) (2006) 253–262
25. Guerraoui, R., Kapalka, M., Vitek, J.: STMBench7: a benchmark for software transactional memory. SIGOPS Oper. Syst. Rev. **41**(3) (2007) 315–324