Replicação Parcial com Memória Transacional Distribuída

João A. Silva, Tiago M. Vale, João M. Lourenço, e Hervé Paulino*

CITI — Departamento de Informática,
Universidade Nova de Lisboa, Portugal
{jaa.silva,t.vale}@campus.fct.unl.pt
{joao.lourenco,herve.paulino}@fct.unl.pt

Resumo Os sistemas de memória transacional distribuída atuais recorrem essencialmente à distribuição ou à replicação total para distribuir os seus dados pelos múltiplos nós do sistema. No entanto, estas estratégias de replicação de dados apresentam limitações. A distribuição não oferece tolerância a falhas e a replicação total limita a capacidade de armazenamento do sistema. Nesse contexto, a replicação parcial de dados surge como uma solução intermédia, que combina o melhor das duas anteriores com o intuito de mitigar as suas desvantagens. Esta estratégia tem sido explorada no contexto das bases de dados distribuídas, mas tem sido pouco abordada no contexto da memória transacional e, tanto quanto sabemos, nunca antes tinha sido incorporada num sistema de memória transacional distribuída para uma linguagem de propósito geral. Assim, neste artigo propomos e avaliamos uma infraestrutura para replicação parcial de dados para programas Java *bytecode*, que foi desenvolvida com base num sistema já existente de memória transacional distribuída. A modularidade da infraestrutura que apresentamos permite a implementação de múltiplos algoritmos e, por conseguinte, avaliar em que contextos de utilização (*workloads*, número de nós, etc.) a replicação parcial se apresenta como uma alternativa viável a outras estratégias de replicação de dados.

Palavras-chave: Replicação parcial; Memória transacional distribuída; Controlo de concorrência; Java *bytecode*

1 Introdução

A memória transacional (MT) apresenta-se como uma alternativa de alto nível para controlo de concorrência que aplica no contexto da gestão de memória partilhada o conceito de *transação*, muito utilizado nas bases de dados. Uma transação é uma sequência de operações que executam segundo uma semântica de

* Este trabalho foi parcialmente financiado pela Fundação para a Ciência e Tecnologia (FCT/MCTES), no contexto do projeto de investigação PTDC/EIA-EIA/113613/2009, do PEst-OE/EEI/UI0527/2011 e da bolsa de investigação SFRH/BD/84497/2012.

tudo ou nada, i.e., ou todas as operações executam com sucesso e como se fossem uma única ação indivisível e instantânea, ou nenhuma executa. Assim, quando utiliza MT, o programador pode encapsular os acessos à memória partilhada dentro de transações, delegando o controlo de concorrência e gestão de conflitos no sistema de execução. Uma das características da MT é ser inerentemente otimista. Uma vez que múltiplas transações podem executar concorrentemente e só no fim é verificada a existência de conflitos, existe maior concorrência quando diferentes transações modificam partes disjuntas da memória partilhada.

Ainda no contexto da MT, a necessidade de endereçar questões como a escalabilidade, fiabilidade e a tolerância a falhas, levou ao desenvolvimento de algoritmos e infraestruturas que usam este paradigma aplicado ao contexto distribuído. No entanto, estas infraestruturas são poucas e ou implementam uma estratégia de distribuição pura de dados [8,11] ou uma estratégia de replicação total [3,4,15]. Adicionalmente, a maioria destas infraestruturas foram construídas de modo a permitir exclusivamente a estratégia de replicação de dados para a qual foram desenhadas, tornando difícil a implementação, experimentação e comparação com estratégias alternativas.

Tanto a distribuição pura como a replicação total têm as suas desvantagens. A distribuição não oferece tolerância a falhas e a replicação total limita a capacidade de armazenamento total do sistema. No entanto, é também possível seguir uma estratégia intermédia de *replicação parcial* [1], onde cada nó do sistema apenas replica um subconjunto dos dados. Na replicação parcial, os dados são *particionados* de forma a serem distribuídos pelos vários nós do sistema, mas cada partição é replicada em vários nós. Assim, é possível aumentar a capacidade de armazenamento total do sistema, fazendo um melhor aproveitamento dos seus recursos, ao mesmo tempo que se garante alguma tolerância a falhas.

Uma ideia subjacente ao conceito de replicação parcial é que nem todos os nós têm de participar na confirmação de uma transação. Uma transação só tem de ser enviada para o conjunto de nós que replicam algum dos dados acedidos por essa mesma transação. Tal resulta em menos mensagens enviadas, sobrecarregando menos a rede, e o custo de coordenação entre nós é menor, potenciando a escalabilidade do sistema. Isto faz com que a replicação parcial seja ideal para explorar a localidade dos dados.

A replicação parcial tem sido explorada maioritariamente no contexto das bases de dados relacionais [12,13,14] e, mais recentemente, também no contexto das bases de dados não relacionais, em específico *key/value data stores*. Já foram propostos algoritmos [9,10] que podem ser aplicados a sistemas de memória transacional distribuída (MTD) mas, tanto quanto sabemos, o sistema que propomos e avaliamos neste artigo é o primeiro que endereça MTD com replicação parcial no contexto de uma linguagem de propósito geral. Com este suporte pretendemos abrir caminho para a realização de estudos sobre a viabilidade do modelo de replicação parcial no contexto muito específico da MT. Nessa medida, estendemos a TribuDSTM [15], uma infraestrutura modular para MTD, com esse mesmo suporte, abrindo caminho para a realização dos supramencionados estudos.

2 TribuDSTM

Para suportar MTD é necessário endereçar alguns desafios, tais como onde guardar e como transferir os dados replicados, e como validar as transações num contexto distribuído. A estratégia passa pela utilização de uma camada de comunicação que permita a interação entre os nós do sistema, sobre a qual se constrói uma segunda camada para a gestão da distribuição dos dados por esses mesmos nós e, por último, é indispensável uma camada que se encarregue do controlo de concorrência local em cada nó.

O desenvolvimento deste tipo de sistemas cresce, muitas vezes, a partir de infraestruturas de MT já existentes, às quais são acrescentados os novos componentes necessários. A TribuDSTM [15] é um exemplo de um sistema desses baseando-se na infraestrutura de MT TribuSTM [5].

2.1 TribuSTM

Todos os algoritmos de MT associam algum tipo de informação à memória gerida (*metadados*), cuja natureza varia consoante o algoritmo, como por exemplo trincos, estampilhas temporais, listas de versões, etc. Uma forma de guardar os metadados é colocando-os numa estrutura de dados externa, que faz um mapeamento entre as posições de memória e os respetivos metadados (estratégia externa de colocação de metadados). Normalmente, esse mapeamento é suportado por uma tabela de dispersão que, por questões de desempenho, não trata colisões, levando a que duas posições de memória possam ser mapeadas para um mesmo metadado. Esta opção é válida para algoritmos cujos metadados não tenham uma ligação forte com as posições de memória associadas, como por exemplo o algoritmo TL2 [6], cujos metadados guardam trincos. No entanto, quando os metadados dependem das posições de memória associadas, e.g., a lista de versões associada a cada posição de memória como acontece na JVSTM [2], a estratégia externa não é suficiente, uma vez que é necessário haver uma relação 1–1 (um-para-um) entre os metadados e as posições de memória.

A Deuce [7] é uma infraestrutura de MT para a linguagem Java que permite implementar vários algoritmos de MT, destacando-se pelo seu baixo nível de intrusão para o programador de aplicações. Apenas é necessário anotar com `@Atomic` os métodos que deverão ser executados como transações, que depois são transformados utilizando reescrita de *bytecode*. No entanto, a Deuce apenas permite o uso da estratégia externa de colocação de metadados.

A TribuSTM estende a Deuce possibilitando a implementação de algoritmos de MT com a estratégia interna (ou *in-place*) de colocação de metadados. Uma vez que se trata de Java, os metadados são injetados (*inlined*) nas classes (através de reescrita de *bytecode*), juntando-se aos campos a que estão associados, permitindo uma relação de 1–1 entre os metadados e os campos dos objetos.

2.2 Colocar o D em TribuDSTM

Para dar resposta ao problema da comparabilidade de estratégias de distribuição e de replicação de dados mencionado na Secção 1, Vale et al. propôs a Tri-

buDSTM [15], uma infraestrutura de MTD para a linguagem Java, caracterizada pela sua modularidade, eficiência e interface não intrusiva para o programador de aplicações. A TribuDSTM permite múltiplas implementações alternativas para os vários componentes que a constituem, que podem depois ser combinados de forma a criar variantes específicas.

A TribuDSTM utiliza a TribuSTM para o controlo da concorrência local em cada nó do sistema, e implementa dois componentes adicionais: um gestor de distribuição (GD) e um sistema de comunicação em grupo (SCG). O GD implementa um sistema de memória distribuída e/ou partilhada de acordo com a estratégia de replicação desejada (distribuição pura, replicação total ou replicação parcial) recorrendo a metadados de distribuição[1], bem como os protocolos de suporte à execução de transações distribuídas. O SCG encapsula as primitivas de comunicação necessárias à implementação do GD, fornecendo-lhes uma interface uniforme, independente do sistema de comunicação concreto utilizado.

Vale et al., em [15], realizou uma implementação concreta do GD para ambientes de replicação total. A confirmação de transações distribuídas segue um protocolo baseado em certificação sem votação, que faz uso de uma primitiva de comunicação de difusão atómica para impor uma ordem total sobre as transações. Mas, devido à sua modularidade, a infraestrutura pode ser estendida para permitir a implementação de outras estratégias de replicação de dados.

Por sua vez, o trabalho apresentado neste artigo implementa o GD tendo como alvo ambientes de replicação parcial (e total, uma vez que replicação total é um caso particular da replicação parcial) e adiciona alguns novos subcomponentes que serão descritos posteriormente, na Secção 3.

3 Suporte para Replicação Parcial com a TribuDSTM

A distribuição pura consiste em *dividir* os dados pelos vários nós do sistema. Esta estratégia maximiza a capacidade de armazenamento total do sistema, uma vez que a quantidade total de dados possível de armazenar é a soma dos recursos de todos os nós. Mas a falha de um único nó põe em causa a fiabilidade do sistema, pois implica a perda de uma parte dos dados. Por sua vez, a replicação total consiste em *replicar* os dados por todos os nó do sistema, de forma a que cada nó possua uma cópia integral dos dados. Esta maximiza a capacidade de sobrevivência dos dados perante falhas, mas a capacidade de armazenamento total do sistema fica limitada à capacidade do nó com menos recursos.

Neste trabalho consideramos uma solução intermédia – a replicação parcial – onde cada nó apenas replica um subconjunto dos dados. A aplicação desta estratégia de replicação implica que os nós do sistema sejam divididos em *grupos* e os dados sejam particionados entre os grupos, de modo a garantir que, no mesmo grupo, todos os nós possuem os mesmos dados. No entanto, isto não impede que existam dados replicados em vários grupos (nem obriga a que os grupos sejam suportados por conjuntos de nós disjuntos).

[1] Diferentes dos metadados referidos na Secção 2.1, que são metadados transacionais.

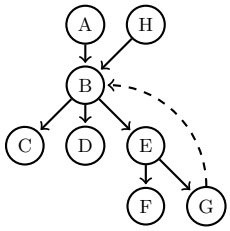


Figura 1: Exemplo de grafo da *heap*.

Uma transação só é confirmada pelo conjunto de nós que replicam algum dos dados acedidos por essa mesma transação, o que resulta num menor custo de coordenação quando comparado com uma estratégia de replicação total.

Consequentemente, uma das vantagens da replicação parcial face à total é a escalabilidade. Propagar cada transação para todos os nós pode tornar-se impraticável para um elevado número de nós e, também, para sistemas que originam um número elevado de transações. Além disso, a replicação parcial pode tirar partido de acesso localizado e reduzir o número de mensagens trocadas.

Os protocolos de replicação parcial dividem-se, essencialmente, em dois grupos: *genuínos*, onde para cada transação T, apenas os nós que possuem dados acedidos por T participam na sua certificação; e *não genuínos*, onde para cada transação T, todos os nós guardam informação sobre T, mesmo que não possuam dados acedidos por T. Os protocolos não genuínos vão contra um dos objetivos da replicação parcial, pois obrigam a que todos os nós participem na certificação de todas as transações.

3.1 Modelo de Programação

Uma vez que a replicação parcial tem sido explorada pelas bases de dados, é natural que estas funcionem como fonte de inspiração para quando pretendemos incorporar uma estratégia de replicação parcial num sistema de MTD para uma linguagem de propósito geral. No entanto, o conceito de replicação parcial tal como é usado pelas bases de dados necessita de ser adaptado para o novo contexto de uma linguagem de propósito geral. Que dados devem ser replicados parcialmente? Qualquer objeto da aplicação pode ser replicado parcialmente? Um objeto pode estar num grupo e referenciar outro objeto que está replicado num grupo diferente? Estas e outras questões serão endereçadas de seguida.

Na *Java Virtual Machine* (JVM), a *heap* pode ser vista como um grafo orientado, onde cada nó corresponde a um objeto que, por sua vez, pode ter referências para outros objetos, como ilustrado na Fig. 1. As folhas do grafo representam tipos básicos e todos os outros nós representam algum objeto ou referência para outra variável. Na Fig. 1, o nó B pode ser visto como um objeto que tem três campos, em que dois são tipos básicos e o outro (nó E) é uma referência para outro objeto que, por sua vez, tem dois campos que são de tipos básicos. Os nós

```
class Node<T> {
  private Node next;
  @Partial
  private T value;
  ...
}
```

(a) Classe `Node`.

```
class List<T> {
    protected Node<T> head;
    public List() {
       this.head = new Node<T>(null, null);
    }
    ...
}
```

(b) Classe `List`.

Figura 2: Exemplo de utilização da anotação @Partial.

A e H podem ser vistos com duas variáveis que referenciam o mesmo objeto B (situação de *aliasing*).

Seguindo o espírito do trabalho anterior de Vale et al. [15], decidimos manter a API pública desta infraestrutura o menos intrusiva possível para o programador. Assim, acrescentámos apenas uma nova anotação @Partial a ser adicionada aos programas. A Fig. 2 exemplifica a aplicação desta anotação para um caso simples de uma lista ligada. Ao adicionar ao campo de uma classe a anotação @Partial, o programador indica que tudo o que está a jusante dessa referência, no grafo da *heap*, deve ser replicado apenas num único grupo, i.e., deve ser replicado parcialmente. Por omissão, tudo o que não esteja explicitamente anotado será replicado totalmente. Assim, oferecemos um modelo de programação flexível, que possibilita ao programador ajustar o nível de distribuição/replicação às características específicas da sua aplicação com recurso à anotação @Partial.

No entanto, a utilização desta anotação carece de algum cuidado. Uma vez que os dados estão distribuídos, aceder a alguns dados pode implicar comunicação com um nó remoto. Este é um custo incontornável da distribuição. No exemplo da Fig. 2, se a anotação @Partial estivesse na variável `next`, o iterar sobre a lista implicaria uma leitura possivelmente remota por cada posição iterada, mesmo que não se inspecionasse o seu valor, comprometendo seriamente o desempenho da aplicação.

A semântica desta anotação é também inspirada nos sistemas de bases de dados com replicação parcial. Nestes, as tabelas existem em todos os nós e apenas a informação que armazenam é que se encontra parcialmente replicada. O mesmo acontece com as *data stores*, onde a estrutura que organiza os dados também se encontra em todos os nós e os dados estão replicados parcialmente. Numa linguagem de propósito geral, a ideia passa por replicar totalmente a estrutura que organiza os dados e replicar parcialmente apenas os dados concretos. No exemplo da Fig. 2, a estrutura que organiza os dados é representada pelos nós da lista ligada (classe `Node`) que estão replicados em todos os nós do sistema, e os dados que estão apenas replicados parcialmente são representados pela variável `value`. Deste modo, o número de leituras remotas resultantes do percorrer da lista é limitado pelo que queremos realmente consultar.

Uma limitação deste modelo é a proibição da existência de arcos no grafo da *heap*, que criem ciclos para nós a montante da anotação @Partial. No caso do

exemplo ilustrado na Fig. 1, se o nó E estivesse anotado com @Partial, o arco a tracejado não poderia existir, uma vez que ao anotar o nó E, o programador está a declarar que os objetos representados pelos nós E, F e G devem ser replicados parcialmente e que, por oposição, tudo o que no grafo esteja a montante deverá ser replicado totalmente. Ao existir uma referência do nó G para o nó B, intuitivamente significa que o nó B deveria ser replicado parcial e totalmente ao mesmo tempo, o que não faz sentido. Por outro lado, se a anotação @Partial estivesse apenas no nó B, o arco a tracejado já poderia existir, uma vez que tudo o que está para jusante no grafo seria replicado parcialmente. Sendo esta uma limitação, achamos que não é uma limitação muito grave. Seguindo a semântica explicada em cima, normalmente a aplicação da distribuição, .i.e., a aplicação da anotação @Partial, é feita ao nível das estruturas de dados. Estas, na maioria das vezes, são implementadas uma única vez, por alguém que conhece o sistema, podendo posteriormente ser utilizadas de forma transparente pelo utilizador comum.

3.2 Metadados e Memória Partilhada e Distribuída

Por questões de balanceamento de carga, disponibilidade e/ou tolerância a falhas, os dados de uma aplicação podem estar distribuídos ou replicados, total ou parcialmente. A gestão dessa distribuição é conseguida através da associação de informação extra — metadados de distribuição — às localizações de memória distribuída. Em [15], Vale et al. tinha como alvo ambientes de replicação total. Assim, cada localização de memória (ou objeto) tinha de ser resolvida de forma unívoca e determinista por todos os nós que compõem o sistema. Para tal, basta associar um identificador único a cada localização de memória.

Metadados. Em ambientes de replicação parcial, não basta existir um identificador único associado a cada objeto. Uma vez que os objetos estão replicados parcialmente, cada objeto é replicado apenas por um subconjunto dos nós do sistema, por isso pode ocorrer o caso em que um nó pretende realizar uma operação sobre um objeto para o qual não existe uma cópia local. Nesse caso, esse nó tem de conseguir identificar o objeto como sendo remoto e obter toda a informação necessária para comunicar com um nó que o replique.

Na nossa implementação, os metadados de distribuição passaram a conter um identificador global e único e, também, a caracterização do grupo no qual esse objeto se encontra replicado. Mais especificamente, passam a conter os endereços dos nós que replicam esse objeto. Assim, se um nó fizer uma leitura transacional a um objeto que não está replicado localmente, é feito um pedido de leitura a todos os nós que replicam o objeto.

Para que se possa inferir diretamente a partir dos metadados de um dado objeto se este se encontra parcial ou totalmente replicado, existe um grupo que representa todos os nós do sistema, o grupo `ALL`.

Grupos e Dados. Em ambientes de replicação parcial, tanto o particionamento dos dados como o agrupamento dos nós são de grande importância, uma vez

que têm grandes implicações no desempenho das aplicações. Um nó que esteja constantemente a ler dados que não replica é forçado a fazer leituras remotas, o que leva a um desempenho fraco. E, se pensarmos num sistema composto por vários *clusters*, um grupo composto por nós de *clusters* diferentes vai aumentar o tempo de confirmação das transações para valores indesejáveis.

Como referimos na Secção 3, em ambientes de replicação parcial, os nós do sistema são divididos em grupos que por sua vez replicam um subconjunto dos dados do sistema. Essa divisão em grupos tem de seguir algum algoritmo/estratégia. Para tal é necessário algo que encapsule estratégias de decisão quanto ao agrupamento dos nós. Com esse intuito, criámos um novo sub-componente que faz isso mesmo, o *particionador de grupos* (PG). Tendo conhecimento dos nós que existem no sistema e do número de grupos que devem existir, o PG devolve um agrupamento segundo a estratégia que implementa. De momento temos implementadas apenas duas estratégias simples: particionamento aleatório e circular (*round robin*), sendo os nomes auto-explicativos.

Existe também um problema quando se cria um novo objeto. Quando a aplicação cria um novo objeto, dentro de uma transação, onde é que esse objeto fica replicado? Um objeto replicado parcialmente só é replicado em alguns nós. Quais? Em que grupos? Para tomar essa decisão é necessário, mais uma vez, algo que encapsule estratégias de decisão quanto ao particionamento dos dados. Na nossa infraestrutura, o sub-componente que tem esse papel é o *particionador de dados* (PD). De momento temos implementadas apenas três estratégias simples: particionamento aleatório, circular (*round robin*) e local. A última diferencia-se das restantes por estabelecer que os objetos são replicados no grupo do nó que os criou, portanto estabelece afinidade entre o nó que executou a transação que gerou o objeto em causa e a localização final do mesmo. As duas primeiras estratégias conseguem manter os dados bem balanceados entre os grupos devido à sua natureza aleatória/circular. A terceira estratégia consegue o mesmo apenas se a quantidade de transações executadas por cada grupo for relativamente igual.

3.3 Transações Distribuídas

Em ambientes de replicação total, os protocolos baseados em certificação mostram-se interessantes por não necessitarem de sincronização entre os nós durante a execução de transações. Estes protocolos são simples e de fácil implementação, e acima de tudo, requerem poucos passos de comunicação.

No entanto, não é fácil suportar replicação parcial a partir de protocolos de replicação total. Os protocolos baseados em certificação baseiam-se em primitivas de comunicação, que garantem que todos os nós recebem as transações pela mesma ordem, funcionando como um autómato finito. No caso da replicação parcial esta solução não funciona. Por exemplo, consideremos um sistema composto por três nós ($N1$, $N2$ e $N3$). O nó $N1$ replica os objetos B e C, $N2$ replica A e B e $N3$ replica A e C. Consideremos ainda que existem duas transações $T1$ e $T2$ a ser executadas concorrentemente em $N1$ e $N2$, respetivamente. A transação $T1$ modifica B e $T2$ modifica A e B. No final, $N1$ e $N2$ certificam ambas as transações mas, $N3$ (como replica apenas A e C) só certifica $T2$. Se a ordem

total for $T1$ e $T2$, $N1$ e $N2$ confirmam $T1$ e cancelam $T2$ mas $N3$ confirma $T2$, modificando o valor de A, o que torna os dados incoerentes.

Assim, os nós não podem confirmar transações sozinhos. Têm de acordar entre todos que transações são confirmadas e que transações são canceladas. É necessário haver uma votação. Seguindo este pensamento, foram propostos vários protocolos de confirmação de transações, para replicação parcial, baseados no protocolo de confirmação em duas fases (2PC). Primeiro, pelas bases de dados distribuídas [13,14], e mais recentemente, no contexto da MTD [9,10].

Graças à modularidade da TribuDSTM é possível implementar vários protocolos de confirmação de transações distribuídas. Assim, e uma vez que existem protocolos desenhados para o contexto específico da MTD, implementámos o GD da TribuDSTM com o protocolo proposto em [9], chamado SCORe.

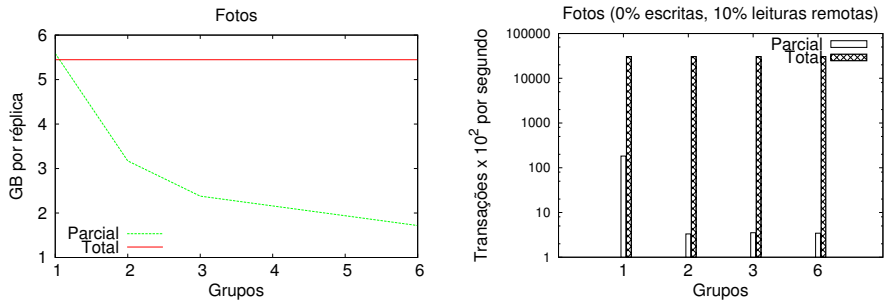
SCORe. O SCORe [9] é um protocolo genuíno para replicação parcial, assim, processa transações envolvendo apenas os nós de mantêm uma cópia dos dados acedidos pelas transações. O protocolo faz uso de um esquema de controlo de concorrência multi-versão distribuído que é associado a um esquema de sincronização com um relógio lógico, oferecendo *1-copy-serializability* (1CS) como modelo de consistência. Este protocolo, quando a transação chega à fase de confirmação, corre um 2PC clássico sobreposto com um protocolo de acordo distribuído que permite ordenar totalmente as transações em todos os nós, fazendo com que as transações sejam aplicadas pela mesma ordem.

4 Resultados Experimentais

Os testes foram efetuados num sistema com seis computadores com $2\times$ Quad-Core AMD Opteron 2376, cada um a 2.3 Ghz, com 4×512 KB *cache* L2 e com 8 GB de RAM. Os computadores executam Linux 2.6.26-2-amd64, Debian 5.0.10, e estão ligados em rede via Ethernet Gigabit privada. A versão da linguagem Java utilizada foi OpenJDK 6. Na nossa infraestrutura foram utilizadas duas configurações. A configuração para replicação parcial utiliza ao nível da camada de MT um algoritmo multi-versão, como apresentado em [9]. O protocolo de confirmação de transações distribuídas utilizado foi o SCORe (Secção 3.3). A configuração para replicação total utiliza TL2 e certificação sem votação. Ao nível do SCG foi utilizado o Spread 4.2[2] em ambas as configurações.

Apresentamos agora resultados para um micro-teste por nós criado. Este teste tenta representar um programa de colagem de fotografias. Existem conjuntos de fotografias que estão divididos por categorias e o programa lê várias dessas fotografias arranjando-as em colagens. O teste é composto por quatro tipos diferentes de transações, nomeadamente inserções, remoções, leituras locais e leituras remotas. O teste foi configurado para um tamanho de seis categorias com 128 fotografias cada. Cada fotografia tem um tamanho que pode variar entre 3 a 12 MB. Com replicação parcial, se existir apenas um grupo (com 6 nós),

[2] `http://www.spread.org`



(a) Consumo máximo médio de memória em cada réplica do sistema.

(b) Desempenho do sistema com 10% de leituras remotas.

Figura 3: Avaliação da infraestrutura para replicação parcial *vs.* total. Utilizando os mecanismos de replicação parcial, com 1 grupo obtém-se replicação total e com 6 grupos distribuição pura.

todas as 6 categorias ficam replicadas nesse grupo (o equivalente a replicação total); se existirem 2 grupos, cada grupo replica 3 categorias, e assim sucessivamente. No noutro extremo temos 6 grupos, cada um replicando uma categoria, o que equivale a distribuição pura (uma vez que cada grupo tem apenas um nó).

A Fig. 3a mostra o uso máximo de memória, em média, por cada nó (eixo das ordenadas), usando a infraestrutura com replicação total e com replicação parcial, variando o número de grupos (eixo das abcissas). Naturalmente, quantos mais grupos existem, menos dados cada grupo tem de replicar, o que faz com que, por exemplo, com 6 grupos cada nó use menos cerca de 3 GB de memória que um nó em replicação total. Quando passamos de um para dois grupos, cada nó passa a replicar apenas metade dos dados, como podemos ver no gráfico da Fig. 3a, onde o consumo de memória, por nó, diminui de cerca de $5,5$ GB para 3 GB.

A Fig. 3b mostra-nos o número de transações por segundo (eixo das ordenadas) variando o número de grupos (eixo das abcissas). Para este teste foi configurada uma taxa de leituras remotas de 10%. Uma leitura remota, no contexto deste teste, significa que um nó pretende aceder a uma fotografia que não replica, o que o obriga a fazer um pedido de leitura aos nós do grupo que replica essa fotografia. As leituras remotas acontecem apenas quando existe mais do que um grupo.

A configuração do sistema com um grupo permite-nos avaliar o *overhead* da infraestrutura e protocolo de replicação parcial num ambiente de replicação total. Como mostra a figura, as transações por segundo com replicação parcial descem da ordem do milhão (com a infraestrutura para replicação total) para dezenas de milhar. Isto deve-se a uma maior complexidade das operações de leitura (porque o protocolo assim o exige). Com a configuração para replicação parcial as operações de leitura efetuam testes adicionais para verificar relógios lógicos, a localidade dos dados e bloqueiam até se verificarem certas condições.

Ao aumentar o número de grupos, verificamos que o desempenho caí para uma ordem de centenas de transações por segundo. Isto deve-se, maioritariamente, a leituras remotas que, durante a nossa experimentação, atingiram latências até 300 ms.

5 Trabalho Relacionado

A replicação parcial de dados começou por ser explorada no contexto das bases de dados distribuídas [1], como uma maneira de explorar outras alternativas e tentar ultrapassar as limitações das outras estratégias de replicação de dados. Por isso, a área da replicação de bases de dados é algo rica em protocolos de consistência de dados para replicação parcial. Das várias propostas [12,13,14], destacamos o P-Store [13], que foi o primeiro a apresentar protocolos genuínos para replicação parcial de dados, oferecendo 1CS como modelo de consistência. Este difere do SCORe na medida em que requer que as transações de leitura sofram uma confirmação distribuída e faz uso de uma primitiva de difusão atómica seletiva para impor uma ordem total às transações.

No contexto da MT, a replicação parcial tem sido alvo de alguma investigação muito recente. Existem duas propostas de protocolos [9,10]. O GMU [10] foi a primeira proposta de um protocolo genuíno para replicação parcial a garantir que transações de leitura nunca são abortadas ou forçadas a uma confirmação distribuída. Este faz uso de um esquema de controlo de concorrência multi-versão que depende de um mecanismo de sincronização baseado em relógios vetor, e oferece *extended update serializability* (EUS) como modelo de consistência. O SCORe [9] é em muito semelhante ao GMU. Este passa a usar um mecanismo de sincronização baseado num relógio lógico e oferece 1CS como modelo de consistência.

De entre as infraestruturas de MTD existentes [3,4,8,11] nenhuma implementa (ou permite implementar) replicação parcial, uma vez que foram construídas especificamente para a estratégia de replicação de dados para a qual foram desenhadas. Das diferenças entre estas infraestruturas e a nossa realçamos a modularidade da nossa infraestrutura (que, neste trabalho, passa a suportar replicação parcial) e, também, a sua não intrusão para o programador de aplicações (necessitando de apenas usar as anotações `@Atomic` e `@Partial`).

6 Conclusões e Trabalho Futuro

A infraestrutura apresentada neste artigo é, tanto quanto sabemos, a primeira a incorporar o conceito de replicação parcial de dados num sistema de MTD para uma linguagem de propósito geral, mais especificamente a linguagem Java. Esta infraestrutura oferece uma interface não intrusiva para o programador de aplicações e permite a integração de diferentes algoritmos de MT e protocolos de confirmação de transações distribuídas. A nova anotação `@Partial` oferece um modelo de programação flexível, permitindo ao programador afinar o nível de distribuição/replicação às especificidades de cada aplicação.

No seu estado atual, a infraestrutura permite a implementação de diferentes estratégias de particionamento de grupos e de dados em conjunto com os diferentes algoritmos de MT e protocolos de confirmação. Direções futuras, no seguimento desta contribuição, incluem a exploração da localidade dos dados, otimizações de modo a melhorar o desempenho, bem como uma experimentação mais exaustiva que nos permita tirar conclusões mais detalhadas.

Referências

1. Gustavo Alonso. Partial database replication and group communication primitives (extended abstract). In *European Research Seminar on Advances in Distributed Systems (ERSADS)*, 1997.
2. João Cachopo and António Rito-Silva. Versioned boxes as the basis for memory transactions. *Sci. Comput. Program.*, 63(2):172–185, December 2006.
3. Nuno Carvalho, Paolo Romano, and Luis Rodrigues. A generic framework for replicated software transactional memories. In *IEEE International Symposium on Network Computing and Applications*, pages 271–274, 2011.
4. Maria Couceiro, Paolo Romano, Nuno Carvalho, and Luís Rodrigues. D2stm: Dependable distributed software transactional memory. In *IEEE Pacific Rim International Symposium on Dependable Computing*, 2009.
5. Ricardo J. Dias, Tiago M. Vale, and João M. Lourenço. Efficient support for in-place metadata in transactional memory. In *Euro-Par*, 2012.
6. Dave Dice, Ori Shalev, and Nir Shavit. Transactional locking ii. In *international conference on Distributed Computing*, pages 194–208, 2006.
7. G. Korland, N. Shavit, and P. Felber. Deuce: Noninvasive software transactional memory in java. *Transactions on HiPEAC*, 5(2), 2010.
8. Christos Kotselidis, Mohammad Ansari, Kim Jarvis, Mikel Luján, Chris Kirkham, and Ian Watson. Distm: A software transactional memory framework for clusters. In *International Conference on Parallel Processing*, pages 51–58, 2008.
9. Sebastiano Peluso, Paolo Romano, and Francesco Quaglia. Score: A scalable one-copy serializable partial replication protocol. In *Middleware 2012*. 2012.
10. Sebastiano Peluso, Pedro Ruivo, Paolo Romano, Francesco Quaglia, and Luis Rodrigues. When scalability meets consistency: Genuine multiversion update-serializable partial data replication. In *International Conference on Distributed Computing Systems*, pages 455–465, 2012.
11. Mohamed M. Saad and Binoy Ravindran. Hyflow: a high performance distributed software transactional memory framework. In *international symposium on High performance distributed computing*, pages 265–266, 2011.
12. Nicolas Schiper, Rodrigo Schmidt, and Fernando Pedone. Optimistic algorithms for partial database replication. In *Principles of Distributed Systems*, volume 4305, pages 81–93. 2006.
13. Nicolas Schiper, Pierre Sutra, and Fernando Pedone. P-store: Genuine partial replication in wide area networks. In *Symposium on Reliable Distributed Systems*, pages 214–224, 2010.
14. António Sousa, Rui Oliveira, Francisco Moura, and Fernando Pedone. Partial replication in the database state machine. In *International Symposium on Network Computing and Applications*, 2001.
15. Tiago M. Vale, Ricardo J. Dias, and João M. Lourenço. Uma infraestrutura para suporte de memória transacional distribuída. In *INForum Simpósio de Informática*, 2012.