PARTIALSTM: Replicação Parcial de Memória Transaccional

Pedro Miguel Pereira Ruivo
pedro.ruivo@ist.utl.pt

Instituto Superior Técnico
(Orientador: Professor Luís Rodrigues)

Resumo O problema da partilha de dados e do controlo de concorrência no acesso a dados partilhados tem sido bastante estudado na literatura. O uso explícito de trincos por parte do programador, acarreta alguns problemas no desenvolvimento de programas concorrentes. O aparecimento da memória transaccional veio dar resposta a estes problemas ao tornarem o controlo de concorrência implícito. Os sistemas de memória transaccional, para além de simplificarem o controlo de concorrência, permitem desenvolver aplicações com requisitos de disponibilidade e escalabilidade. Estes sistemas combinam a utilização da distribuição com a replicação. No entanto, a distribuição só oferece escalabilidade enquanto a replicação só oferece disponibilidade. Assim, a replicação parcial surgiu de modo a juntar as vantagens de cada uma destas técnicas. Este relatório propõe alternativas para suportar replicação parcial e apresenta sugestões para as desenvolver e avaliar.

1 Introdução

O problema da partilha de dados e do controlo de concorrência no acesso a dados partilhados tem sido bastante estudado na literatura e é central ao desenvolvimento da maioria das aplicações que usamos diariamente. Aplicações bancárias *on-line* ou redes sociais são apenas dois exemplos de aplicações onde dados podem ser actualizados e lidos concorrentemente por vários processos. Este problema ganhou uma redobrada importância com o advento das arquitecturas de processadores multi-núcleo (*multi-core*). Ao contrário do que aconteceu no passado, em que a velocidade dos computadores aumentava[1] praticamente ao ritmo da lei de Moore[2], hoje em dia, o aumento da velocidade de execução tem de ser obtido através da paralelização de código, de forma a explorar os vários núcleos disponíveis nos processadores modernos. Para ser efectiva, esta paralelização deve poder suportar paralelismo de grão fino.

O problema do controlo de concorrência consiste em evitar que dois processos acedam simultaneamente aos mesmos dados. Uma zona do código que acede a dados partilhados, e que deve ser executada em exclusão mútua, é tipicamente

[1] 4 MHz em 1981; 100 MHz em 1995; 3 GHz em 2002; 5 GHz em 2010
[2] http://www.computerhistory.org/semiconductor/timeline/1965-Moore.html

designada por secção crítica (a execução do código de uma secção crítica por dois ou mais processos, sem qualquer mecanismo de controlo de concorrência, pode deixar os dados num estado incoerente). Os mecanismos de controlo de concorrência têm como objectivo resolver este problema e podem ser explícitos ou implícitos.

O controlo de concorrência explícito recorre a mecanismos de sincronização de baixo nível como os trincos (*locks*). Infelizmente, estes mecanismos são susceptíveis de serem incorrectamente aplicados pelos programadores, resultando em situações de interbloqueio (*deadlock*) ou míngua (*starvation*).

Uma das abstracções mais poderosas para evitar a utilização explícita de mecanismos de controlo de concorrência de baixo nível é a noção de transacção. Originalmente proposta no contexto das aplicações de bases de dados, uma transacção é definida como uma sequência de operações de leitura e escrita em dados partilhados, que deve ser executada de forma isolada, sem entrelaçamento com outras transacções que acedam aos mesmos dados. Para além disso, todas as operações de uma transacção são aplicadas de forma atómica, isto é, ou todas as actualizações são aplicadas ou nenhuma o é (neste último caso, diz-se que a transacção cancelou). Note-se que, para assegurar o isolamento, o sistema de suporte à execução das transacções pode recorrer ao uso de trincos; no entanto, estes são geridos automaticamente, sem a intervenção directa do programador.

Neste projecto estamos interessados em desenvolver técnicas que suportem o desenvolvimento de sistemas concorrentes, suportando transacções em sistemas distribuídos e tolerantes a faltas. Nestes sistemas, o controlo de concorrência necessita ser integrado com os mecanismos de gestão da distribuição e replicação, para assegurar que os utilizadores continuam a obter dados correctos, apesar de existirem diversas cópias dos itens em diferentes nós do sistema. Quando se considera o recurso à distribuição e replicação, existem duas alternativas que representam utilizações extremas de cada uma destas técnicas:

- A primeira consiste em dividir os dados pelas diversas máquinas do sistema sem recorrer a replicação. Neste caso, maximiza-se a capacidade de armazenamento, uma vez que a quantidade total de dados que é possível armazenar advém da soma dos recursos de todas as máquinas. A desvantagem desta aproximação é que a falha de uma única máquina causa a perda de uma parte dos dados.
- A segunda consiste em replicar os dados em todas as máquinas, de forma a que cada nó do sistema possua uma cópia integral dos dados. Neste caso, maximiza-se o potencial de sobrevivência dos dados perante falhas. A desvantagem desta aproximação é que os recursos existentes não são explorados para aumentar a capacidade de armazenamento do sistema, que fica limitada à capacidade do nó com menos recursos no sistema.

Neste trabalho consideramos uma solução intermédia, em que os dados são particionados de forma a serem distribuídos por vários nós do sistema, mas em que cada partição é mantida de forma replicada por vários nós. Este tipo de arquitectura suporta aquilo que se designa por *replicação parcial*. Num sistema com

estas características, uma transacção envolve necessariamente vários nós, mais precisamente, os nós que possuem réplicas dos dados acedidos pela transacção. Como transacções diferentes podem aceder a dados diferentes, os conjuntos de nós envolvidos em cada transacção não são necessariamente os mesmos, embora possam também não ser disjuntos. Assegurar o controlo de concorrência e a gestão da replicação neste cenário e de forma eficiente é um desafio significativo.

O remanescente deste relatório está organizado da seguinte forma. A Secção 2 sumariza os objectivos e os resultados esperados do nosso trabalho. Na Secção 3 expõe-se o trabalho relacionado. A Secção 4 descreve a solução proposta e a Secção 5 descreve as técnicas previstas para avaliar os resultados. A Secção 6 inclui o planeamento para trabalho futuro. Finalmente, a Secção 7 conclui o relatório.

2 Objectivos

Este trabalho aborda o problema da replicação parcial em memória transaccional. Mais precisamente:

> *Objectivos:* O principal objectivo deste trabalho é analisar, desenvolver e avaliar protocolos de replicação parcial, no contexto de sistemas de memória transaccional.

Este trabalho começa com uma panorâmica sobre o tema em questão, a qual é apresentada na secção seguinte. As várias arquitecturas propostas para suportar distribuição e replicação de dados são analisadas, identificando-se as suas vantagens e desvantagens.

Este relatório propõe dois algoritmos para suportar replicação parcial em sistemas de memória transaccional distribuída. Um deles é semelhante a um dos algoritmos descritos no trabalho relacionado. O segundo algoritmo propõe algumas modificações de modo a melhorar o desempenho em relação ao anterior. Assim, o objectivo é atingir os seguintes resultados:

> *Resultados esperados:* Este trabalho irá (i) produzir a especificação dos algoritmos; (ii) desenvolver e concretizar os mesmos; e (iii) realizar uma avaliação experimental dos algoritmos desenvolvidos.

3 Trabalho relacionado

De modo a entender os problemas que este trabalho implica, é necessário possuir alguns conhecimentos fundamentais na área de sistemas distribuídos.

Assim, a restante secção está dividida no seguinte modo. Na Subsecção 3.1 é apresentado alguns conceitos sobre memória transaccional. O sistema de gestão de base de dados e de memória transaccional em *software* são muitos semelhantes, pelo que, na Subsecção 3.2, é efectuada uma breve comparação entre ambos. Normalmente, na distribuição e replicação de dados são usadas primitivas de

comunicação em grupo. Assim, na Subsecção 3.3 são introduzidas algumas dessas primitivas, que são necessárias para compreensão dos algoritmos descritos. Nas Subsecções 3.4 e 3.5 são descritos algoritmos que suportam distribuição e replicação de dados, respectivamente. Finalmente, na Subsecção 3.6 são introduzidos os algoritmos referentes à replicação parcial e a Subsecção 3.7 conclui esta secção.

3.1 Memória transaccional

Como foi referido na introdução, a memória transaccional[1,2] pretende simplificar o desenvolvimento de aplicações concorrentes através de uma abstracção designada por transacção. Usando esta abstracção, o programador só necessita de delimitar as secções críticas do código, através de primitivas que iniciam e terminam uma transacção. Por sua vez, o ambiente de execução fica responsável por executar os mecanismos necessários para realizar o controlo de concorrência, recorrendo, se necessário, a trincos que são adquiridos e libertados sem intervenção do programador [3].

Para além disso, todas as operações de uma transacção são executadas de forma atómica. Quando a transacção termina, é executado um procedimento com o objectivo de realizar a confirmação (*commit*) da transacção. Caso não se verifiquem erros ou violações do modelo de coerência dos dados devido ao acesso concorrente aos mesmos, a transacção é confirmada e todas as escritas realizadas no seu contexto se tornam visíveis. Caso contrário, a transacção é cancelada e todas as actualizações aos dados são descartadas.

As transacções são usadas de forma generalizada no contexto dos sistemas de gestão de bases da dados, onde asseguram as garantias ACID que, de forma informal, podem ser descritas da seguinte forma:

- *Atomicidade:* garante que todas ou nenhumas operações são executadas;
- *Coerência:* a execução das transacções não viola a coerência dos dados;
- *Isolamento*: a execução de uma transacção não é afectada pela execução concorrente de outras transacções;
- *Durabilidade:* as modificações, depois de confirmadas, tornam-se persistentes.

Os sistemas de memória transaccional são inspirados nos sistemas de bases de dados embora com algumas características específicas. Em muitos casos, a persistência dos dados não é um requisito. As transacções podem ser bastante curtas e o modelo de coerência de dados pode ser mais exigente, uma vez que as mesmas transacções que cancelam devem ser protegidas de observar dados incoerentes (uma propriedade designada por opacidade[4]).

De forma a realizar o controlo de concorrência, é geralmente necessário recolher informação acerca de quais os dados acedidos por uma transacção (designado por *dataset*). O *dataset* está dividido no conjunto de dados que são lidos (conjunto de leitura ou *readset*) e no conjunto de dados que são actualizados (conjunto de escrita ou *writeset*). Estes dois conjuntos não são geralmente

conhecidos à partida, caso em que as transacções se designam por dinâmicas. Quando o *dataset* é conhecido à partida as transacções designam-se por estáticas.

O controlo de concorrência pode seguir uma abordagem optimista ou pessimista. Na abordagem optimista[1], a gestão de conflitos é adiada o mais possível. Quando começa uma transacção, assume-se que a transacção poderá ser confirmada e que não existirão conflitos com outras transacções. Nos acessos à memória, a transacção não adquire os respectivos trincos e as escritas e leituras são colocadas num histórico local à transacção (*log*). Quando a transacção termina, verifica-se se os dados lidos pela transacção não foram entretanto escritos por outras transacções concorrentes. Caso os dados estejam no mesmo estado, isto significa que não ocorreu nenhum acesso concorrente e a transacção pode ser confirmada, actualizando-se o conjunto de escrita de forma atómica. Caso tenham ocorrido conflitos, a transacção é cancelada, perdendo-se o trabalho realizado durante a sua execução. Esta é a principal desvantagem desta abordagem.

Na abordagem pessimista[2], a gestão dos conflitos é efectuada o mais cedo possível. Associa-se um trinco a cada objecto de memória e quando a transacção faz um acesso à memória, o trinco correspondente é adquirido. Os conflitos são detectados mais cedo e o trabalho perdido é menor. No entanto, um modelo de execução pessimista pode originar interbloqueios entre as transacções. Assim, o sistema deve prevenir ou detectar (e resolver) essas situações. Em transacções estáticas é possível prevenir a ocorrência de interbloqueios se os trincos forem adquiridos por ordem determinista. No entanto, em transacções dinâmicas, é necessário executar algoritmos de detecção e/ou prevenção de interbloqueio. Para mais pormenores sobre a prevenção e detecção de interbloqueios, sugere-se a consulta de [5], páginas 546 a 548.

A escolha entre a abordagem optimista e pessimista depende do tipo de sistema. Na abordagem optimista, tem-se o custo de fazer um única validação no final da transacção, enquanto na abordagem pessimista tem-se o custo de adquirir/libertar os trincos juntamente com o de prevenir/detectar os interbloqueios. Desta forma, em sistemas com poucos conflitos, uma abordagem optimista é mais vantajosa.

Os mecanismos de concorrência [2] podem ser implementados em *hardware* (HTM[6,7,8,9,10]), *software* (STM[11,12,13,14,15]), ou de uma forma híbrida (HyTM[16,17,18,19]). Os mecanismos baseados em *hardware* são mais rápidos, mas não são flexíveis e possuem escolhas fixas ou limitadas de implementação. O oposto acontece com os mecanismos por *software*, que são mais lentos, mas oferecem uma maior flexibilidade.

3.2 Sistemas de gestão de base de dados e memória transaccional em *software*

Os sistemas de gestão de base de dados (DBMS) são conjuntos de programas que permitem armazenar, modificar e extrair grandes quantidades de dados a partir de uma base de dados. Uma base de dados é um conjunto estruturado de registos. A forma como os registos estão estruturados é denominado o modelo da base de dados. Uma transacção é a unidade de trabalho efectuada no DBMS.

A memória transaccional em *software* (STM) e os DBMS possuem várias características em comum, nomeadamente:

- ambos suportam a abstracção de transacção;
- ambos controlam de forma automática o acesso concorrente aos dados.

No entanto, existem também diferenças significativas entre estes dois tipos de sistemas, nomeadamente:

- as STM não possuem o requisito de assegurar a persistência dos dados;
- as transacções numa STM podem conter código arbitrário que não está restrito pela sintaxe da linguagem usada para aceder a bases de dados, o SQL;
- o número de instruções de uma transacção em sistemas STM pode ser significativamente menor do que o número de instruções de uma transacção num DBMS;
- as transacções numa STM não são executadas num ambiente confinado, pelo que a leitura de valores incoerentes pode gerar desvios na sua execução tais como, a geração de excepções de hardware ou ciclos infinitos, que podem tornar difícil ou mesmo impossível cancelar as mesmas. Por esta razão, os modelos de coerência para STMs são, por vezes, mais restritos que os modelos de coerência para DBMS. Enquanto nas bases de dados, o modelo de coerência mais forte é a serializabilidade[20], nas STM tenta-se também assegurar a *opacidade*[4], que impede qualquer transacção de ver resultados incoerentes (mesmo as transacções que estão condenadas a cancelar).

3.3 Comunicação em grupo

Os protocolos de comunicação em grupo oferecem comunicação ponto a multiponto. O objectivo é cada processo do grupo receber uma cópia das mensagens enviadas para o grupo, com garantias de entrega. Estas garantias asseguram que todos os membros do grupo recebem o mesmo conjunto de mensagens e todos concordam na ordem de entrega das mensagens à aplicação. Comunicação em grupo é um paradigma poderoso que pode ser usado para manter a coerência de várias réplicas, na construção de uma STM distribuída e/ou replicada.

Atomic Broadcast

Atomic Broadcast[21] (ABcast), também conhecido por *Total Order Broadcast*, é uma primitiva de comunicação em grupo que permite o envio de mensagens para todos os processos, com a garantia que todos concordam no conjunto de mensagens entregues e na ordem pela qual são entregues. ABcast pode ser definido através das primitivas `TO-broadcast(m)` e `TO-deliver(m)`, onde `m` é uma mensagem, e possui as seguintes propriedades:

- *Ordem:* Se um processo entrega a mensagem A antes da mensagem B, então todos os processos entregam a mensagem A antes da mensagem B;

- *Terminação:* Se um processo correcto envia uma mensagem, então todos os processos correctos, alguma vez, entregam a mensagem;
- *Atomicidade:* Se um processo entrega uma mensagem, então todos os processos correctos, alguma vez, entregam a mensagem.

Optimistic Atomic Broadcast

O ABcast é importante para construir aplicações distribuídas mas, infelizmente, a sua concretização pode ser muito dispendiosa. Isto deve-se ao número de passos de comunicação e às mensagens trocadas que são necessárias para implementá-lo, o que resulta num atraso significativo antes da mensagem ser entregue.

O *Optimistic Atomic Broadcast* (OABcast) tem como objectivo minimizar os problemas de latência da primitiva anterior. Neste serviço, um membro do grupo entrega à aplicação uma mensagem assim que a recebe (entrega espontânea). Deste modo, a aplicação pode começar por processar a mensagem, em paralelo com a coordenação necessária para determinar a ordem total. No final da fase de coordenação, a mensagem (ou um identificador da mensagem) é entregue na sua ordem final (entrega final), que corresponde à ordem total. O OABcast possui portanto uma primitiva adicional, `Opt-deliver(m)`, que estima uma ordem final em cada processo (ordem espontânea), sendo caracterizado pelas seguintes propriedades[22,23]:

- *Terminação:* Se um processo correcto envia uma mensagem m, então, alguma vez, entrega m na ordem espontânea;
- *Acordo global:* Se um processo correcto entrega uma mensagem m na ordem espontânea, então todos os processos correctos, alguma vez, entregam m na ordem espontânea;
- *Acordo local:* Se um processo correcto entrega uma mensagem m na ordem espontânea, então entrega m na ordem final;
- *Ordem global:* Se dois processos entregam m e m' na ordem final, então fazem-no pela mesma ordem;
- *Ordem local:* Se um processo entrega na ordem final uma mensagem m, então entregou m na ordem espontânea, anteriormente.

A Figura 1 mostra a vantagem do OABcast em relação ao ABcast. Se a ordem espontânea for igual à ordem final, então consegue-se ganhar tempo para processamento, indicado por *tempo ganho* na figura. Verifica-se que a probabilidade da ordem final ser igual à ordem espontânea é muito elevada quando todos os nós encontram-se ligados entre si no mesmo segmento de uma rede local [24].

Reliable Broadcast

ABcast e OABcast acima definidos garantem a ordem total na entrega de mensagens. No entanto, existem situações onde só é necessário garantir que uma cópia da mensagem seja recebida por todos os processos. O *Reliable Broadcast*[25]

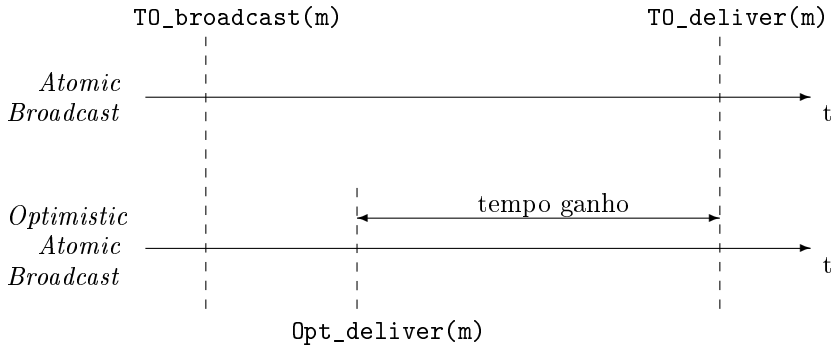

Figura 1. Diferença entre *Atomic Broadcast* e *Optimistic Atomic Broadcast*

(RBcast) é uma primitiva semelhante ao ABcast, mas que não garante a ordem total da entrega de mensagens. Além disso, a sua implementação é mais simples, porque não é necessário a fase de coordenação que determina a ordem total das mensagens.

O RBcast é definido pelas primitivas `R-broadcast(m)` e `R-deliver(m)` e é caracterizado pelas seguintes propriedades:

- *Validade:* Se um processo correcto envia uma mensagem m, então, alguma vez, entrega m;
- *Acordo:* Se um processo correcto entrega uma mensagem m, então, alguma vez, todos os processos correctos entregam a mensagem m;
- *Integridade:* Para cada mensagem m, cada processo entrega m uma única vez, se esta foi previamente enviada.

Atomic Multicast

Atomic Multicast (AMcast) é uma variação do ABcast. Este permite o envio de mensagens para um subconjunto dos processos, enquanto o ABcast envia para todos os processos. AMcast também garante que todos os processos concordam na ordem de entrega das mensagens [26,27,28]. É definido pelas primitivas `A-multicast(m)` e `A-deliver(m)` e possui as seguintes propriedades[28]:

- *Integridade:* Para qualquer processo s e qualquer mensagem m, s entrega m no máximo uma vez, se e só se $s \in m.dst$[3] e m foi previamente enviada;
- *Validade:* Se um processo correcto s envia uma mensagem m, então, alguma vez, todos os processo correctos $s' \in m.dst$ entregam m;
- *Acordo global:* Se um processo s entrega uma mensagem m, então, alguma vez, todos os processos correctos $s' \in m.dst$ entregam m;

[3] $m.dst$ indica o conjunto de réplicas a quem a mensagem m se destina

– *Ordem prefixa uniforme:* Para qualquer duas mensagens m e m', e para quaisquer dois processos s e s' tais que $\{s, s'\} \subseteq m.dst \cap m'.dst$, se s entrega m e s' entrega m', então s entrega m' antes de m ou s' entrega m antes de m'

3.4 Distribuição de dados

O aparecimento da arquitectura multi-núcleo veio promover um maior interesse na memória transaccional por *software* (STM). No entanto, as STM estão a começar a enfrentar requisitos de escalabilidade. Neste contexto, definimos escalabilidade[29] como a capacidade de um sistema se adaptar ao aumento do número de nós. Como as STM e os sistemas de gestão de base de dados (DBMS) partilham o conceito de transacção, as STM começaram a adoptar soluções existentes para DBMS [30].

A distribuição consiste em dividir os dados pelos diversos nós do sistema sem recorrer à replicação. Neste caso, aumenta-se a quantidade de dados que é possível armazenar e divide-se a carga pelos nós do sistema. Normalmente, uma transacção não necessita de aceder a todos os dados. Assim, não é necessário que seja certificada por todos os nós. Isto permite que o sistema seja mais escalável mas, se um dos nós falhar, pode levar à perda dos dados armazenados nesse nó.

A maioria dos protocolos para suportar distribuição de dados foca-se em arquitecturas de memória partilhada, deixando o domínio dos *clusters* inexplorado. O comportamento destes protocolos é diferente em *clusters*, devido ao custo da troca de uma mensagem entre nós [31]. Em *clusters* existem dois objectivos importantes: (i) reduzir (ou eliminar) as operações remotas (comunicação); (ii) assegurar que a implementação da STM permite agregar mensagens e explorar o acesso localizado.

Um exemplo de uma STM distribuída é o DiSTM[31]. No DiSTM são apresentados três protocolos para obter coerência de dados. Todos assumem a existência de um nó especial — o mestre — com quem todos comunicam e que possui um comportamento diferente dos restantes. Esses protocolos são o *Transactional Coherence and Consistency* (TCC[32]) e dois protocolos baseados no conceito de *leases*[33].

No TCC cada transacção adquire um *ticket*, mecanismo através do qual se define uma ordem global das transacções. A validação é efectuada no final da transacção, recorrendo à distribuição do *writeset* (WS) e do *readset* (RS). Cada nó vai comparar o WS e o RS das transacções concorrentes e, caso seja detectado algum conflito, a decisão de cancelar depende da ordem de serialização, isto é, a transacção com o *ticket* mais recente deve ser cancelada. A coerência é garantida através de uma aproximação pessimista no mestre. No caso de a transacção ser confirmada, as actualizações são tornadas públicas, quer localmente, quer globalmente no mestre, que possui todos os dados. Cada nó pode ter uma cópia (*cache*) de dados de outros nós. O mestre invalida estas cópias, cada vez que os dados são alterados.

Nos outros dois protocolos, cada transacção adquire um *lease*. O papel do *lease* é serializar as confirmações das transacções, evitando o custo de distribuir

o *dataset*. Quando uma transacção possui o *lease* e não foi cancelada pode confirmar as suas modificações, pois é garantido que mais ninguém vai confirmar ou certificar outras transacções. As modificações são tornadas públicas no mestre, o que pode provocar o cancelamento de outras transacções. A diferença entre os dois protocolos baseados em *leases* é que um deles possui um único *lease*, enquanto o outro possui múltiplos *leases*. O uso de múltiplos *leases* tem a vantagem de confirmar várias transacções em simultâneo, mas com o custo de uma validação extra efectuada no mestre, onde é verificado que não existe conflitos com transacções que possuem outros *leases*.

3.5 Replicação de dados

A distribuição permite que um sistema escale, uma vez que a execução das transacções pode ser distribuída pelos nós. No entanto, com a falha de um dos nós pode ocorrer perda dos dados. Em alguns sistemas, a disponibilidade dos dados é um factor importante. A replicação é uma técnica que recorre à criação e manutenção de várias cópias dos dados em múltiplos nós, com o objectivo de aumentar a disponibilidade, tolerância a faltas e fiabilidade de um sistema [34]. Além disso, permite aumentar o paralelismo do sistema, embora acarrete custos de sincronização entre réplicas, para garantir a coerência dos dados.

Na restante subsecção descreve-se algoritmos de replicação de dados. Começa-se por descrever o algoritmo primário – secundário e de seguida o algoritmo máquina de estados. De seguida, são apresentados algoritmos onde a transacção é executada numa única réplica e no final é certificada por todas as réplicas (secção Algoritmos com certificação) e, finalmente, são apresentados algoritmos em que todas as réplicas executam a transacção (secção Algoritmos sem certificação).

Primário – Secundário

Nesta técnica, também conhecida por replicação passiva[34], só umas das réplicas — o nó primário — processa os pedidos de escrita dos clientes e propaga as actualizações para as outras réplicas — os nós secundários. As actualizações podem ser propagadas de modo síncrono ou assíncrono. No primeiro caso, o primário fica bloqueado até obter uma confirmação dos secundários, indicando que as actualizações foram aplicadas. Os secundários mantêm os dados coerentes e as operações de leitura nos secundários devolvem o valor mais recente. No modo assíncrono, as actualizações são adiadas. Neste caso, as operações de leitura nos secundários podem devolver um valor antigo.

A principal vantagem desta técnica é a sua simplicidade e o facto de envolver menos processos redundantes [35]. Além disso, todas as operações de escritas são dirigidas ao primário, o que torna fácil a sua ordenação e serialização. Normalmente, a detecção de falha do primário é efectuada através de *timeouts*. Isto origina uma desvantagem caso o primário fique sobrelotado (e por isso, demora demasiado tempo na resposta a pedidos). O atraso no primário pode ser detectado como uma falha, levando um dos secundários a assumir o seu lugar. Caso aconteça, dois primários podem estar activos originando incoerência nos dados.

Máquina de estados

Esta abordagem envolve todas as réplicas no processamento dos pedidos do cliente. Nesta técnica, também conhecida por replicação activa[36], o cliente envia o seu pedido para todas as réplicas. O pedido é processado e no final as réplicas respondem ao cliente. O cliente pode esperar pela primeira, por uma maioria ou por todas as respostas. A principal vantagem desta técnica é que a falha é totalmente transparente para o cliente [35]. No entanto, esta técnica necessita de uma primitiva de comunicação que garanta que todas as réplicas recebem os pedidos pela mesma ordem (por exemplo, ABcast), e assim, toda a complexidade fica escondida pela primitiva de comunicação. Outra desvantagem é que esta técnica só funciona para pedidos determinísticos.

Comparando replicação activa e passiva, podemos concluir o seguinte:

- A replicação activa usa mais recursos computacionais do que a passiva. Isto deve-se ao facto de todas as réplicas processarem os pedidos do cliente;
- Em casos de falha, a replicação passiva tem uma maior latência. O cliente necessita de repetir o pedido para outra réplica;
- Replicação activa necessita de operações determinísticas. Todas as réplicas necessitam de chegar ao mesmo resultado de modo a garantir a coerência dos dados. Isto já não acontece no caso da replicação passiva.

Algoritmos com certificação

As duas alternativas anteriores restringem os nós onde os pedidos podem ser efectuados. Com o uso de algoritmos de certificação, um pedido pode ser efectuado em qualquer nó garantindo, no final, a coerência dos dados. Estes algoritmos têm como objectivo garantir que todas as réplicas concordam no resultado final de uma transacção.

O *Two-Phase-Commit*[37] (2PC) é o algoritmo mais simples e mais usado. Neste algoritmo, o nó que submete a transacção actua como coordenador e os restantes como participantes. O algoritmo tem duas fases. Na primeira fase, o coordenador dá a conhecer a transacção e todos os participantes enviam-lhe o seu voto, indicando se é ou não possível confirmar a transacção. Na segunda fase, completam a transacção, após recepção de uma mensagem de confirmação ou de cancelamento enviada pelo coordenador.

O 2PC possui a desvantagem de bloquear caso o coordenador falhe e os participantes estejam no estado não decidido, isto é, ainda não tenham recebido a resposta do coordenador. Neste estado, todos os participantes mantêm os trincos nos dados até receberem a resposta final do coordenador, o que pode impedir a certificação de novas transacções.

O *Three-Phase-Commit*[37] (3PC) adiciona mais uma fase no coordenador, de modo a evitar a ocorrência da situação acima definida.

Os algoritmos em [38] e [39] são baseados no modelo de actualização diferida (*deferred update*) e usam ABcast para comunicação entre réplicas. De acordo com o modelo de actualização diferida, as transacções são processadas localmente

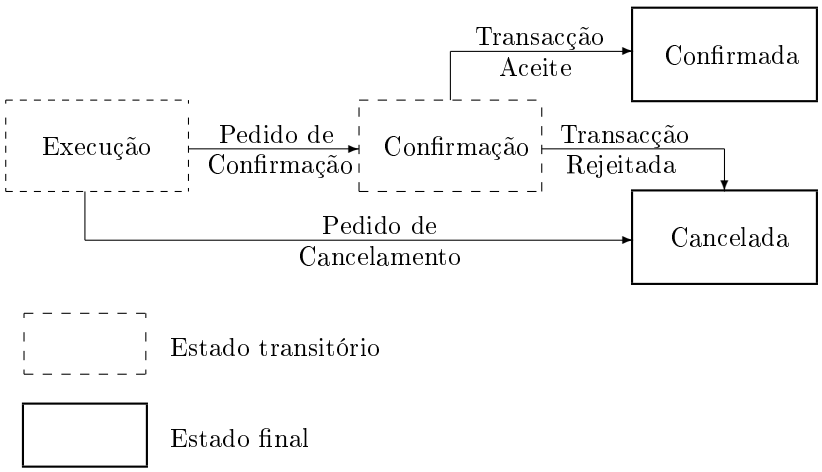

Figura 2. Estados de um transacção no modelo de actualização diferida

numa réplica e, em tempo de confirmação, são enviadas para as outras réplicas. Este modelo tem as seguintes vantagens[40]:

- Melhor desempenho, através da recolha e propagação de várias modificações juntas, e por executar a transacção numa única réplica, possivelmente perto do cliente;
- Melhor suporte para tolerância a faltas, por simplificar a recuperação das réplicas;
- Baixa taxa de interbloqueios, pela eliminação de interbloqueios distribuídos.

No entanto, este modelo também possui algumas desvantagens, entre as quais, um aumento na taxa de cancelamentos, devido à ausência de sincronização durante a execução da transacção.

Neste modelo, uma transacção passa por estados bem definidos [39] (ver Figura 2):

- *Execução:* A transacção começa neste estado, onde são executadas as operações de leitura e escrita. Quando a transacção pede a confirmação, passa para o estado *Confirmação* e é enviada para as outras réplicas;
- *Confirmação:* Uma transacção recebida encontra-se neste estado e mantém-se nele até o seu resultado ser conhecido. A transacção é certificada quando é recebida;
- *Confirmada/Cancelada:* Estado final da transacção.

Em [38] uma transacção T pode ser confirmada (isto é, passar para o estado *Confirmada*) se para qualquer transacção T' que está no estado *Confirmada*, T' precede[4] T ($T' \rightarrow T$) ou T' não modificou nenhum item lido por T.

Em [39] é proposta uma estratégia para ordenar as transacções com o objectivo de obter uma menor taxa de cancelamentos. Assim, uma transacção T pode ser confirmada na posição p, se as seguintes condições se verificarem:

- Para cada transacção T' antes da posição p, T' precede[5] T ($T' \rightarrow T$) ou não modificou itens lidos por T
- Para cada transacção T'' depois da posição p, T'' não precede T ($T'' \nrightarrow T$) ou T'' não modificou itens lidos por T, e T não modificou itens lidos por T''

Ao contrário das transacções em DBMS, as transacções em STM não possuem custos de acesso ao disco, nem custos de interpretação e optimização de SQL [30]. Como consequência, os custos da comunicação em grupo são mais visíveis e formam o principal ponto de estrangulamento na certificação de transacções.

Um exemplo de um sistema STM é o D2STM[41]. Com o objectivo de diminuir os custos da comunicação, D2STM apresenta uma nova técnica de certificação denominada *Bloom Filter Certification*. Esta utiliza uma técnica de codificação recorrendo a *bloom filters*[42] que permite reduzir o tamanho do RS. No entanto, introduz um custo (aumento da taxa de cancelamento) provocado pelos falsos positivos dessa técnica. A taxa de cancelamento é inversamente proporcional ao tamanho final da codificação, o que permite que esta taxa seja controlada.

O modelo de execução é semelhante aos protocolos de replicação descritos anteriormente. As transacções só de leitura executam-se localmente e são sempre confirmadas, pois a JVSTM[43], STM que está na base deste sistema, garante que lêem de um *snapshot* válido. As restantes transacções são certificadas localmente, antes de dar início ao protocolo de certificação. Se passar na certificação local, a transacção é enviada através do ABcast com o seu RS codificado. A validação é feita verificando se os valores lidos não foram modificados por transacções concorrentes.

Em [44] é apresentado uma nova abordagem baseada em *leases*. O algoritmo é denominado de *Asynchronous Lease Certification*. Os *leases* são usados por uma réplica de modo a conceder-lhe privilégios temporários na gestão de um subconjunto dos dados. Esta abordagem consegue reduzir o tempo da fase de confirmação e protege as transacções de serem repetidamente canceladas devido a conflitos remotos. Tal como na D2STM[41] é usada a JVSTM[43] que garante que as transacções de leitura nunca são canceladas.

[4] $T' \rightarrow T$ se T' foi confirmada (estado *Confirmada*) antes de T começar a sua execução (estado *Execução*)

[5] $T' \rightarrow T$ se (i) se T e T' executam-se no mesmo nó, então T' passou ao estado *Confirmação* antes de T; e (ii) se T e T' executam-se em nós diferentes, s_i e s_j respectivamente, então T' entrou no estado *Confirmada* em s_i antes de T entrar no estado *Confirmação* em s_i.

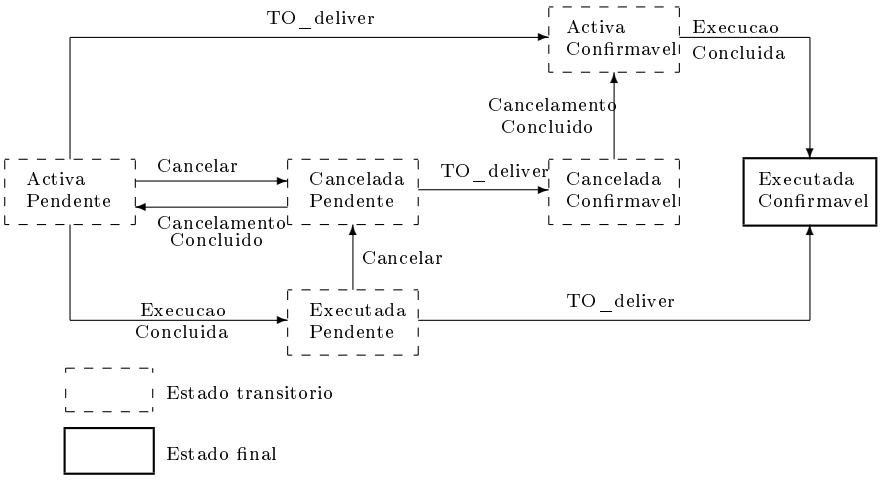

Figura 3. Estados das transacções

De forma análoga aos sistemas anteriores, as transacções são executadas localmente e no final dá-se início ao algoritmo de coordenação entre réplicas. Primeiro, é feita uma certificação local e, só no caso de não haver conflitos, é que se tentam adquirir os *leases* necessários. Quando o nó local já possui todos os *leases* é feita a certificação final, e se não houver conflitos é enviado o WS. Finalmente, depois de aplicar as modificações no nó local, libertam-se os *leases* adquiridos.

No caso de ser necessário re-executar a transacção, os *leases* não são libertados. Assim, caso o conjunto de dados acedidos seja o mesmo, não será necessário voltar a adquirir os *leases*. Caso isto não se verifique, podem ocorrer interbloqueios na aquisição dos novos *leases*, o que pode ser resolvido por um simples mecanismo de detecção de interbloqueios. Note-se que a detecção de interbloqueios pode ser feita localmente e não necessita de nenhuma coordenação entre réplicas.

Algoritmos sem certificação

Em [23] é apresentado um algoritmo que tira vantagem da primitiva OABcast. Ao usar OABcast, pode-se beneficiar da existência de redes onde se verifica a ordenação espontânea das mensagens (por exemplo, as LAN). Nestas redes existe uma grande probabilidade das mensagens serem entregues optimisticamente por ordem total [45]. Segundo resultados experimentais descritos em [23], as mensagem são entregues fora da ordem nos vários nós devido, principalmente, a erros de *buffer overflow* e não a transmissões concorrentes. Este algoritmo usa transacções estáticas que podem estar num estado definido na Figura 3.

Quando um cliente inicia uma transacção, esta é enviada para todas as réplicas. Ao ser recebida na ordem espontânea, os trincos dos dados acedidos são colocados na fila de espera respectiva. Isto é possível pois são usadas transacções estáticas, que tem a vantagem do *dataset* ser conhecido *à-priori*. Cada item que uma réplica possui tem associado uma fila de espera com os trincos. As operações de uma transacção são executada à medida que os seus trincos são adquiridos. Os trincos são adquiridos consoante o tipo do primeiro trinco da fila:

- Se o primeiro trinco da fila é um trinco para escrita, então só esse é libertado;
- Se o primeiro trinco da fila é um trinco para leitura, então são libertados todos os trincos para leitura seguintes, até encontrar um para escrita.

Quando é detectado alguma transacção fora de ordem, as operações já executadas são anuladas e os trincos reposicionados nas filas de espera.

AGGRO[22] é outro algoritmo de replicação de STM baseado em OABcast. Quando uma transacção é iniciada, é enviada através desta primitiva para todas as réplicas. Assim que é recebida na ordem espontânea, a transacção começa logo a sua execução. A detecção de conflitos é verificada cada vez que é escrito ou lido algum valor. Além disso, permite ler dados de transacções que acabaram a sua execução mas que ainda não foram confirmadas. Isto leva a que as transacções sejam ligadas entre si. Se uma delas cancelar, devido a ler valores inválidos ou por ter sido entregue fora de ordem final, pode ocorrer um cancelamento em cadeia. No entanto, em redes com entrega espontânea, esses casos são poucos frequentes e consegue-se um aumento do desempenho do sistema.

3.6 Replicação parcial de dados

Na replicação total, propagar cada transacção para todas as réplicas não é adequado para um elevado número de réplicas nem para sistemas que originam um número elevado de transacções. Ao contrário da replicação total, a replicação parcial pode aumentar o acesso localizado e reduzir o número de mensagens [46]. A replicação parcial combina as vantagens da replicação e da distribuição mas, sem as desvantagens de cada uma delas.

O conceito de replicação parcial refere-se a uma variação da replicação total, na qual uma réplica não possui todos os dados [47]. Cada réplica só guarda um determinado subconjunto de todos os dados da base de dados e esse conjunto pode ter cópias em várias réplicas.

A ideia principal da replicação parcial é que nem todos os nós têm de processar uma transacção. A transacção só tem de ser enviada para o conjunto de réplicas que possuem os dados acedidos pela transacção. Assim, a rede fica menos sobrecarregada e o custo da coordenação entre réplicas é menor [46].

No entanto, não é fácil suportar replicação parcial a partir dos protocolos de replicação total. Normalmente, a replicação total é baseada em primitivas de comunicação em grupo, que garantem que todas as réplicas recebem a transacção pela mesma ordem. Se o funcionamento for semelhante a um autómato finito, esta solução funciona bem. No entanto, esta solução não funciona em replicação

parcial [47]. Por exemplo, considere um sistema composto por três réplicas (S_1, S_2 e S_3). A réplica S_1 contém B e C, S_2 contém A e B e S_3 contém A e C. Considere ainda que existem duas transacções t_1 e t_2 a ser executadas concorrentemente em S_1 e S_2, respectivamente. A transacção t_1 modifica B e t_2 modifica A e B. No final, S_1 e S_2 certificam ambas as transacções mas, S_3 só certifica t_2. Se a ordem total for t_1 e t_2, S_1 e S_2 confirmam t_1 e cancelam t_2 mas S_3 confirma t_2, modificando o valor de A, o que torna os dados incoerentes.

Os protocolos de replicação parcial dividem-se em três grupos[28,25]:

- *Genuínos:* Para cada transacção T, as réplicas que certificam T são aquelas que possuem dados acedidos por T;
- *Quasi-genuínos:* Para cada transacção T, réplicas correctas que não possuem nenhuns dados acedidos por T, não armazenam permanentemente mais do que o identificador de T;
- *Não genuínos:* Para cada transacção T, todas as réplicas guardam informação sobre T, mesmo que não não possuam nenhuns dados acedidos por T.

Os protocolos não genuínos vão contra o objectivo da replicação parcial, pois todas as réplicas têm de se envolver na transacção, o que conduz aos problemas da replicação total.

Em [48] é apresentado um novo algoritmo denominado *Resilient Atomic Commit* (RAC). Neste algoritmo, quando a transacção termina, o seu *dataset* é enviado para todas as réplicas que possuam cópias dos dados acedidos pela transacção. Cada réplica realiza uma certificação local e envia o seu voto (sim ou não) consoante o resultado da certificação. O envio do voto pode ser efectuado de dois modos: (i) cada réplica envia o voto para as restantes réplicas envolvidas; ou (ii) cada réplica envia o voto para uma das réplicas (normalmente a réplica que executou a transacção) e essa réplica indica às restante o resultado da transacção. Independente do modo, no final todas as réplicas chegam à mesma decisão (*confirmada* ou *cancelada*). O RAC possui as seguintes propriedades:

- *Acordo:* Não há dois processos que tomem decisões diferentes;
- *Terminação:* Cada processo alguma vez decide;
- *Validade:* Se um processo decide *confirmada* para T, então para cada item x acedido por T, existe pelo menos um processo que contém x e votou sim para T;
- *Não trivialidade:* Se para cada item x acedido por T existe pelo menos um processo que possui x, votou sim para T e não é suspeito, então todos os participantes correctos alguma vez decidem *confirmada* para T.

Além do RAC, em [48] utiliza-se a primitiva de comunicação OABcast para enviar a transacção para as outras réplicas. Quando a transacção é entregue na ordem espontânea, as réplicas iniciam o algoritmo (RAC). Se a ordem final for diferente da espontânea, repetem-no.

O algoritmo em [25] utiliza uma aproximação diferente. Neste algoritmo, as transacções são enviadas para todas as réplicas através da primitiva RBcast,

que não garante ordem total. Por outro lado, este algoritmo usa a abstracção de consenso de modo a determinar uma ordem de serialização para as transacções.

O problema do consenso[25] pode ser definido do seguinte modo: cada participante propõe um determinado valor e no final todos concordam no valor decidido. Este é caracterizado pelas seguintes propriedades:

- *Integridade:* cada participante decide no máximo uma vez;
- *Acordo:* não há dois participantes com decisões diferentes;
- *Validade:* se um participante decide v, então v foi proposto por algum participante;
- *Terminação:* todos os participantes correctos alguma vez decidem.

A ideia do algoritmo é a seguinte:

1. Quando a transacção termina é enviada através do RBcast para todas as réplicas e, quando é entregue, é colocada numa fila de espera.
2. Enquanto houver transacções na fila, cada réplica envia os votos[6] para cada uma das transacções e, simultaneamente, propõe essa sequência no consenso.
3. Quando o consenso é decidido, a réplica verifica se existe alguma transacção na sequência decidida, tal que o voto ainda não foi enviado. Caso isto se verifique, envia o voto.
4. Cada transacção é certificada pela ordem que aparece na sequência. É verificado se tem todos os votos (bloqueia se faltar algum voto) e que estes são todos *sim*. Se não houver nenhum conflito com transacções que estão antes dela na sequência, a transacção é confirmada.

Os autores em [49] identificam o uso de *1-Copy-Serializibility* como um dos factores que limita escalabilidade de um sistema. Assim, é apresentado um algoritmo quasi-genuíno com critério de consistência *1-Copy-Snapshot-Isolation* (1CSI). Este tem a principal vantagem de não necessitar de enviar o RS, o que reduz o tamanho das mensagens trocadas. Usando 1CSI não ocorrem conflitos *Read-Write*, o que reduz a taxa de cancelamento.

Quando a transacção é iniciada, é atribuído um *timestamp* de início (ST). No final da execução da transacção, esta é enviada para todos os nós. Cada réplica que possua itens acedidos pela transacção efectua a certificação da mesma, mas antes é atribuído um *timestamp* de confirmação (CT). A certificação de T é feita verificando se não existe nenhuma transacção T' tal que $T.WS \cap T'.WS \neq \emptyset \wedge T'.ST \leq T.CT \leq T'.CT$.

P-Store[28] é um algoritmo genuíno. As réplicas do sistema são divididas em grupos e os dados são particionados entre os grupos, de modo a garantir que, no mesmo grupo, todas as réplicas possuem os mesmos dados. No entanto, isto não impede que um item esteja replicado em vários grupos.

O algoritmo funciona do seguinte modo: quando a transacção acaba a sua execução, é enviada uma mensagem através da primitiva AMcast (ver Secção 3.3)

[6] O voto só contém o resultado de certificar a transacção com transacções já confirmadas.

para todas as réplicas que possuem itens acedidos pela transacção. Quando a transacção é entregue, é colocada numa fila. Se a transacção for local, isto é se, só acede a dados dentro de um grupo, cada réplica pode decidir por si, sem necessitar de trocar qualquer informação; caso contrário, diz-se que a transacção é global e todas as réplicas trocam votos entre si para decidir o resultado da transacção.

3.7 Discussão

Nesta secção começa-se por descrever algumas fundamentos sobre a memória transaccional e uma breve comparação entre DBMS e STM. Devido à recente utilização de primitivas de comunicação em grupo para desenvolver algoritmos de replicação/distribuição, é feita uma introdução às mesmas.

Nos algoritmos para suportar distribuição é usado um mecanismo que oferece privilégios a um determinado nó, de modo a poder confirmar as alterações efectuadas. Além disso, ambos os algoritmos permitem que cada nó possua uma *cache* com dados de outros nós, com o objectivo de optimizar a execução de transacções.

Em relação ao algoritmos de replicação, começámos com os algoritmos primário-secundário e máquina de estados. Verifica-se que estes restringem as réplicas onde se podem efectuar os pedidos para executar as transacções, o que facilita na serialização das operações.

De seguida, são apresentados algoritmos que não restringem a réplica onde se efectuam pedidos. Estes dividem-se em dois grupos. No primeiro grupo, o pedido é efectuado numa réplica, que executa a transacção localmente. Precede-se o envio do *dataset* para as restantes réplicas para certificar a transacção. No segundo grupo, o pedido é efectuado numa réplica que envia a transacção para as restantes réplica, com o objectivo de todas as réplicas executarem a transacção. São usadas as propriedades de ordem total das primitivas de comunicação, de modo a obter-se uma ordem de serialização igual em todas as réplicas.

Na replicação parcial, podemos verificar que existe um *tradeoff* entre os passos de comunicação e a genuinidade do algoritmo. Se o algoritmo é genuíno implica uma votação entre as réplicas envolvidas e consequente um maior número de passos de comunicação. Além disso, verifica-se que a concretização deste tipo de algoritmos é mais complexa em relação aos algoritmos para replicação total. Como no anterior, as propriedades de ordem total das primitivas de comunicação oferecem uma ordem de serialização das transacções.

4 Solução proposta

4.1 Arquitectura

Vamos utilizar o *Infinispan*[7] como *framework* de desenvolvimento para o nosso trabalho. O *Infinispan* é uma plataforma de *data grid* de elevada escalabi-

[7] http://www.jboss.org/infinispan

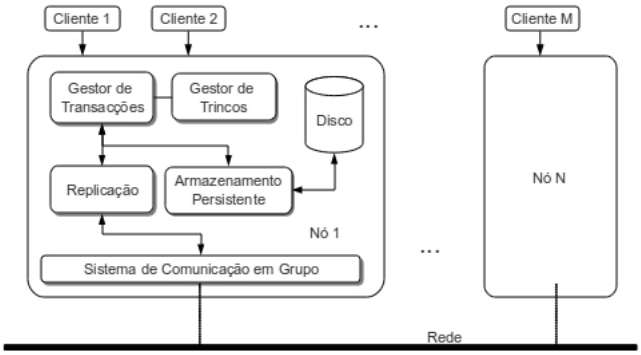

Figura 4. Arquitectura do *Infinispan*

lidade e disponibilidade. Esta permite que os dados sejam replicados ou distribuídos. Usa o 2PC como protocolo para manter a coerência de dados e oferece uma coerência fraca (*read committed* ou *repeatable read*). Além disso, os trincos são mantidos durante a execução da transacção até à sua confirmação (ou até ser cancelada). Isto origina interbloqueios entre transacções. Desta forma, o *Infinispan* pode beneficiar da aplicação de algoritmos de suporte à replicação parcial que permitam oferecer o mesmo modelo de coerência de modo mais escalável.

Na arquitectura do *Infinispan*, ilustrada na Figura 4, cada um dos nós é composto pelos seguintes componentes:

- *Gestor de Transacções*: Este componente é responsável pela execução das transacções, com origem local ou remota.
- *Gestor de Trincos*: Este componente é responsável pela gestão dos trincos que são adquiridos pelas transacções e inclui detecção de interbloqueios. Caso seja detectado um interbloqueio entre duas transacções, cancela-se uma delas.
- *Replicação*: Este componente é responsável por garantir a coerência dos dados entre as várias réplicas. Efectua a certificação das transacções pelo uso do 2PC em conjunto com o *Gestor de Transacções*.
- *Armazenamento Persistente*: Este componente é responsável por garantir o armazenamento e carregamento dos dados de forma persistente (pode ser um disco ou uma base de dados, remota ou local). Se estiver activo, pode funcionar num dos seguintes modos:
  - *Activation/Passivation*: Os dados que um nó possui encontram-se armazenados persistentemente ou em memória. Quando algum item é necessário, é movido para memória (é apagado do disco/base de dados). O contrário acontece quando já não é necessário ter o item em memória;
  - *Load/Store*: Os dados estão em memória e armazenados persistentemente. Quando um item é necessário, é efectuada uma cópia para memória.

A vantagem do primeiro modo é o espaço total de armazenamento. Este é composto pela soma do espaço de armazenamento da memória e do disco/base de dados. No entanto, corre-se o risco de se perder os dados em memória. No segundo, o espaço de armazenamento é igual ao espaço de armazenamento do disco/base de dados (pois, normalmente é maior que o da memória), mas o risco de perder os dados é menor.
- *Sistema de Comunicação em Grupo*: Este componente é responsável por manter informação sobre os membros do grupo (incluindo detecção de falhas) e oferece suporte para a comunicação entre os nós.

Um cliente pode iniciar uma transacção em qualquer réplica do sistema. A transacção é executada localmente. A informação mantida sobre os dados de uma transacção são a chave do item acedido e, para o caso de ser uma modificação, o seu antigo e novo valor. Quando a transacção termina, é enviado o seu WS de modo a ser certificado segundo o funcionamento do protocolo 2PC. A transacção é então confirmada ou cancelada nas réplicas envolvidas.

O módulo de *Sistema de Comunicação em Grupo* fornece o suporte para a comunicação. A ordem total das mensagens não é garantida para as mensagens trocadas pois, por enquanto, não há uma necessidade. A *framework* de comunicação em grupo usada é o *JGroups*[8].

4.2 Algoritmos

Neste trabalho, dois algoritmos de gestão da coerência dos dados serão implementados com o objectivo de melhorar o protocolo de replicação actualmente implementado no *Infinispan* — baseado em 2PC. Este protocolo possui os problemas já definidos anteriormente (ver Secção 3.5, Algoritmos com certificação) e, o modo como gere os trincos, origina interbloqueios entre transacções. Os dois algoritmos aqui propostos têm como principal vantagem não gerar interbloqueios e um deles sobrepõe a comunicação com a execução da transacção. Com a eliminação dos interbloqueios, prevê-se uma diminuição da taxa de cancelamentos e assim, um aumento do desempenho do sistema (isto é, mais transacções processadas por segundo). Ambos os algoritmos vão oferecer a mesma coerência fraca que é oferecida actualmente no *Infinispan*.

Algoritmo 1 – A1

O primeiro algoritmo é muito semelhante ao algoritmo *P-Store*[28], e baseia-se na mesma lógica de organização dos dados. Cada transacção é executada localmente, sem sincronização entre réplicas, e para cada item acedido é feito uma cópia, sobre a qual a transacção trabalha.

Quando a transacção termina a sua execução, o seu WS é enviado (através de AMcast) para as réplicas necessárias, que a certificam. Tal como no *P-Store*, há

[8] http://www.jgroups.org/

duas alternativas possíveis. Se todos os dados acedidos pela transacção pertencem ao mesmo grupo, então só é necessário o envio do WS (através de AMcast) e as réplicas podem imediatamente decidir o resultado da transacção. Caso contrário, é necessário haver uma troca adicional de votos (efectuada através do RBcast) entre as várias réplicas, de modo a chegar a um acordo sobre o resultado final da transacção. Falta referir que as transacções só de leitura nunca cancelam, isto é, são sempre confirmadas. Este algoritmo é *genuíno*, isto é, as réplicas que possuem dados acedidos por uma transacção, são aquelas que recebem e certificam a transacção.

Uma vez que o *Infinispan* não garante serialização mas sim um nível de coerência mais fraco, neste trabalho tomámos a decisão de adaptar a certificação do P-Store, para oferecer um nível de coerência equivalente. Isto irá implicar a alteração da certificação, de modo a apenas enviar o WS, para detectar conflitos entre escritas. Sendo assim, a certificação é efectuada verificando se os valores escritos não foram alterados por transacções concorrentes. Isto será feito de modo a permitir uma comparação justa dos nossos algoritmos com o algoritmo baseado em 2PC, que está actualmente implementado no *Infinispan*.

Algoritmo 2 – A2

O segundo algoritmo irá tentar melhorar o desempenho através da sobreposição da execução da transacção com a comunicação.

Quando uma transacção é iniciada, o seu identificador é enviado (através de AMcast) para todas as réplicas envolvidas. Isto vai fornecer uma ordem pela qual as transacções que possuem conflitos entre si serão certificadas. Quando uma réplica recebe a transacção, esta é colocada numa fila de espera. Uma vez terminada, o seu WS é enviado (através de *Reliable Multicast*, isto é, AMcast sem ordem total, onde a última propriedade não se verifica (ver Secção 3.3)) para as réplicas envolvidas. O envio do WS não necessita de ordem, pois a ordem da certificação das transacções já foi estabelecida inicialmente. As réplicas certificam a transacção mas, só quando esta atingir o início da fila de espera. Este algoritmo é *genuíno*, assumindo que são conhecidas as réplicas que estão envolvidas na transacção.

Na prática, esta hipótese pode ser suportada através de diferentes técnicas: análise de código (manual ou automática), ou técnicas de aprendizagem com base em estatística, as quais prevêem com alguma precisão os padrões de acesso a dados de uma transacção e portanto o conjunto de réplicas correspondentes. Caso a previsão não seja precisa, a correcção do algoritmo não será comprometida. Sempre que as réplicas acedidas numa transacção diferem do previsto, ainda é possível enviar uma nova mensagem (através de AMcast) que actualiza o conjunto de réplicas associadas à transacção.

Uma técnica alternativa mais simples consiste em enviar (através de ABcast) o identificador da transacção para todas as réplicas. O WS continua a ser enviado só para as réplicas com cópias dos dados acedidos pela transacção, enquanto para as restantes réplicas é enviada uma mensagem mais curta, indicando que a transacção deve ser ignorada. Assim, o algoritmo torna-se *quasi-genuíno*,

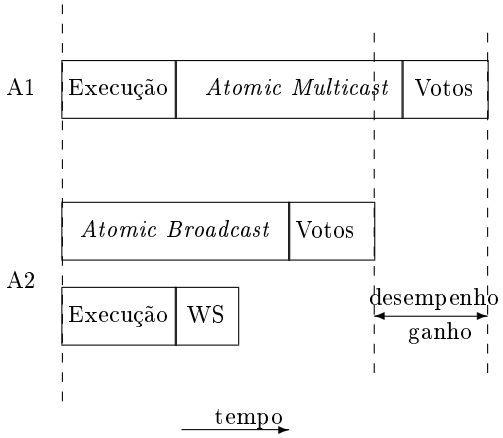

Figura 5. Comparação entre A1 e A2

isto é, as réplicas que não possuem nenhuns dados acedidos pela transacção, só necessitam de guardar o identificador da transacção.

A Figura 5 mostra uma comparação entre os dois algoritmos, A1 e A2. No algoritmo A1 é usado AMcast de modo a propagar as transacções para os grupos de réplicas necessários. Na execução do algoritmo A2, ilustrado na Figura 5, estamos a assumir o pior caso, onde é usado o ABcast. No entanto, este tem a vantagem de sobrepor a comunicação com a execução da transacção.

Comparação com o algoritmo baseado em 2PC

Uma das vantagens dos algoritmos A1 e A2 descritos anteriormente é serem livres de interbloqueios, prevendo-se uma menor contenção de dados e consequentemente uma menor taxa de cancelamentos, em particular com cargas elevadas, por exemplo, um elevado número de transacções de escrita intensiva.

Outra vantagem é a sobreposição da execução da transacção com a comunicação, no caso do algoritmo A2. A comunicação pode tornar-se um ponto de estrangulamento do sistema, que pode ser mitigado ao fazer a sobreposição da comunicação com a execução da transacção.

Com a ajuda da informação da Tabela 1, podemos verificar que ambos os algoritmos possuem o mesmo número de passos de comunicação (2PC: 3; A1: 2 + 1 (AMcast + RBcast); A2: 2 + 1 (ABcast + RBcast)). No entanto, no caso do algoritmo A2, alguns destes passos de comunicação são sobrepostos (ABcast é sobreposto com a execução da transacção e o envio do WS).

A implementação do ABcast no *JGroups* possui um custo de 2 passos de comunicação e tem o seguinte funcionamento: (1) o nó envia (em *unicast*) a mensagem para o sequenciador e (2) o sequenciador envia a mensagem para todas as réplicas (em *multicast*), juntamente com o número de sequência. O *JGroups* possui uma implementação para RBcast, mas não para AMcast.

Protocolo	Passos de comunicação
2PC[37]	3
ABcast[50]	2
AMcast[51]	2
RBcast[25]	1

Tabela 1. Custos mínimos dos diferentes protocolos

4.3 Desafios

Um dos desafios que se colocam durante o desenvolvimento dos algoritmos é o de gerir a complexidade da *framework Infinispan*. A implementação dos algoritmos requer que sejam efectuadas modificações em quase todos os componentes envolvidos na replicação.

Para além disso, de modo a evitar interbloqueios, é necessário proceder à alteração do controlo de concorrência local. A interacção do componente de replicação com o controlo de concorrência local deve ser efectuado de uma tal forma que, se houver conflito entre uma transacção em execução e uma transacção em certificação, seja cancelada a transacção em execução.

5 Metodologias de Avaliação

Este trabalho será integrado com o sistema *Infinispan*. O desempenho dos diferentes algoritmos será determinado através de avaliações experimentais. Depois de obtidos os dados da execução dos testes a cada algoritmo, será feita uma comparação entre o algoritmo original do *Infinispan*, baseado em 2PC, e os algoritmos A1 e A2.

5.1 Métricas

As métricas usadas para avaliar os algoritmos serão os seguintes:

- o número de transacções processadas em cada réplica;
- o número de transacções processadas por segundo pelo sistema;
- o número de transacções canceladas.

O objectivo da replicação parcial é melhorar a escalabilidade de um sistema e em consequência, melhorar o número de transacções que o sistema consegue processar. A primeira métrica indica como a carga foi distribuída pelas réplicas existentes. Espera-se que o número de transacções processadas por cada réplica seja semelhante durante os vários testes, de modo a tornar os restantes resultados mais confiáveis. Ao observar o número de transacções processados por segundo, podemos fazer uma comparação do desempenho de cada algoritmo. Prevê-se um aumento do número de transacções processadas, por parte das nossas soluções, em comparação com o algoritmo baseado em 2PC. Finalmente, o número de transacções canceladas verifica a taxa de cancelamentos existentes em cada um dos algoritmos.

5.2 *Benchmarks*

Actualmente, não existem muitos *benchmarks* disponíveis de modo a avaliar este tipo de sistema. Deste modo, o único *benchmark* que iremos usar será a *Radargun*[9]. O mesmo é usado para testar o desempenho das várias versões do *Infinispan* e constituirá um bom ponto de partida. O principal objectivo é stressar múltiplos nós ao mesmo tempo, com o mesmo padrão de acesso. Este é composto por uma sequência de passos que são executados ao mesmo tempo em todos os nós. São geradas um determinado número de operações de leitura/escrita com base em parâmetros de configuração atribuídos (percentagem de escritas, número de pedidos, etc.). No final, gera e apresenta os resultados obtidos (pedidos por segundos, escritas por segundo, etc.).

No entanto, é também interessante verificar como as soluções propostas se comportam sobre diferentes padrões de carga. Para tal, vamos desenvolver testes, alguns baseados nos casos de teste disponibilizados com a *framework Cache4J*[10]. Os casos de teste desta *framework* possuem uma arquitectura simples, que permite adaptar-se a vários produtos e implementar os nossos testes de uma forma simples.

Finalmente, iremos utilizar os casos de teste disponibilizados para o *Infinispan*, de modo a recolher mais dados. Com isto, iremos comparar os resultados obtidos e gerar a informação necessária para a avaliação deste trabalho.

6 Planeamento de trabalho futuro

O planeamento do trabalho futuro será o seguinte:

- 7 de Janeiro - 31 de Março: Desenho detalhado e implementação da arquitectura proposta, incluindo testes preliminares.
- 1 de Abril - 5 de Maio: Efectuar a avaliação completa dos resultados.
- 6 de Maio - 30 de Maio: Escrever um artigo descrevendo o projecto.
- 31 de Maio - 14 de Junho: Acabar de escrever a dissertação.
- 15 de Junho: Entrega da dissertação de mestrado.

7 Conclusões

Este trabalho aborda o problema do desenvolvimento e concretização de um algoritmo eficiente de replicação parcial para sistemas de memória transaccional. Com a replicação parcial advém os requisitos de escalabilidade e disponibilidade impostos aos sistemas de memória transaccional distribuída actuais. Foi efectuada uma pesquisa sobre sistemas de memória transaccional, algoritmos de coerência de dados para distribuição e replicação. Através da análise efectuada

[9] http://radargun.sourceforge.net/
[10] http://cache4j.sourceforge.net

verifica-se que a replicação parcial é uma boa técnica para oferecer escalabilidade e disponibilidade, pois combina as vantagens da distribuição de dados (escalabilidade) e da replicação de dados (disponibilidade).

Este relatório é concluído com uma descrição da arquitectura que irá ser usada para desenvolver, comparar e validar as soluções propostas.

Agradecimentos

Agradeço ao orientador, professor Luís Rodrigues, a toda a equipa do Grupo de Sistemas Distribuídos, nomeadamente ao Nuno Carvalho, Paolo Romano, Maria Couceiro, José Mocito, Xavier Vilaça, João Fernandes e Nuno Machado, e também aos restantes colegas pelas discussões e comentários produtivos durante a elaboração deste relatório. Este trabalho foi parcialmente financiado pelo projecto ARISTOS (PTDC/EIA-EIA/102496/2008)

Referências

1. Dubrofsky, E.: A survey paper on transactional memory
2. Shriraman, A., Dwarkadas, S., Scott, M.L.: Flexible decoupled transactional memory support. In: Proceedings of the 35th Annual International Symposium on Computer Architecture. ISCA '08, Washington, DC, USA, IEEE Computer Society (2008) 139–150
3. Zhang, J., Chen, W., Tian, X., Zheng, W.: Exploring the emerging applications for transactional memory. In: Proceedings of the 2008 Ninth International Conference on Parallel and Distributed Computing, Applications and Technologies, Washington, DC, USA, IEEE Computer Society (2008) 474–480
4. Guerraoui, R., Kapalka, M.: On the correctness of transactional memory. In: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. PPoPP '08, New York, NY, USA, ACM (2008) 175–184
5. Ramakrishnan, R., Gehrke, J.: Database Management Systems. 2nd edn. McGraw-Hill Higher Education (2000)
6. Ananian, C.S., Asanovic, K., Kuszmaul, B.C., Leiserson, C.E., Lie, S.: Unbounded transactional memory. IEEE Micro **26** (January 2006) 59–69
7. Hammond, L., Wong, V., Chen, M., Carlstrom, B.D., Davis, J.D., Hertzberg, B., Prabhu, M.K., Wijaya, H., Kozyrakis, C., Olukotun, K.: Transactional memory coherence and consistency. In: Proceedings of the 31st annual international symposium on Computer architecture. ISCA '04, Washington, DC, USA, IEEE Computer Society (2004) 102–
8. Herlihy, M., Moss, J.E.B.: Transactional memory: architectural support for lock-free data structures. In: Proceedings of the 20th annual international symposium on Computer architecture. ISCA '93, New York, NY, USA, ACM (1993) 289–300
9. Moore, K., Bobba, J., Moravan, M., Hill, M., Wood, D.: Logtm: log-based transactional memory. In: High-Performance Computer Architecture, 2006. The Twelfth International Symposium on. (2006) 254 – 265
10. Rajwar, R., Herlihy, M., Lai, K.: Virtualizing transactional memory. In: Proceedings of the 32nd annual international symposium on Computer Architecture. ISCA '05, Washington, DC, USA, IEEE Computer Society (2005) 494–505

11. Dice, D., Shalev, O., Shavit, N.: Transactional locking ii. In: In Proc. of the 20th Intl. Symp. on Distributed Computing. (2006)
12. Fraser, K., Harris, T.: Concurrent programming without locks. ACM Trans. Comput. Syst. **25** (May 2007)
13. Herlihy, M., Luchangco, V., Moir, M., Scherer, III, W.N.: Software transactional memory for dynamic-sized data structures. In: Proceedings of the twenty-second annual symposium on Principles of distributed computing. PODC '03, New York, NY, USA, ACM (2003) 92–101
14. Marathe, V.J., Iii, W.N.S., Scott, M.L.: Adaptive software transactional memory. In: In Proc. of the 19th Intl. Symp. on Distributed Computing. (2005) 354–368
15. Saha, B., Adl-Tabatabai, A.R., Hudson, R.L., Minh, C.C., Hertzberg, B.: Mcrt-stm: a high performance software transactional memory system for a multi-core runtime. In: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming. PPoPP '06, New York, NY, USA, ACM (2006) 187–197
16. Damron, P., Fedorova, A., Lev, Y., Luchangco, V., Moir, M., Nussbaum, D.: Hybrid transactional memory. In: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems. ASPLOS-XII, New York, NY, USA, ACM (2006) 336–346
17. Kumar, S., Chu, M., Hughes, C.J., Kundu, P., Nguyen, A.: Hybrid transactional memory. In: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming. PPoPP '06, New York, NY, USA, ACM (2006) 209–220
18. Minh, C.C., Trautmann, M., Chung, J., McDonald, A., Bronson, N., Casper, J., Kozyrakis, C., Olukotun, K.: An effective hybrid transactional memory system with strong isolation guarantees. In: Proceedings of the 34th annual international symposium on Computer architecture. ISCA '07, New York, NY, USA, ACM (2007) 69–80
19. Shriraman, A., Spear, M.F., Hossain, H., Marathe, V.J., Dwarkadas, S., Scott, M.L.: An integrated hardware-software approach to flexible transactional memory. In: Dept. of Computer Science, Univ. of Rochester. (2007)
20. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency control and recovery in database systems. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (1986)
21. Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. (1993) 97–145
22. Palmieri, R., Quaglia, F., Romano, P.: AGGRO: Boosting STM Replication via Aggressively Optimistic Transaction Processing. In: 9th IEEE International Symposium on Network Computing and Applications (NCA), Cambridge, Massachussets, USA, IEEE Computer Society Press (July 2010)
23. Kemme, B., Pedone, F., Alonso, G., Schiper, A., Wiesmann, M.: Using Optimistic Atomic Broadcast in Transaction Processing Systems. IEEE Transactions on Knowledge and Data Engineering **15** (2003) 1018–1032
24. Rodrigues, L., Mocito, J., Carvalho, N.: From spontaneous total order to uniform total order: different degrees of optimistic delivery. In: Proceedings of the 2006 ACM symposium on Applied computing. SAC '06, New York, NY, USA, ACM (2006) 723–727
25. Schiper, N., Schmidt, R., Pedone, F.: Optimistic Algorithms for Partial Database Replication. In: Proceedings of the 10th International Conference on Principles of Distributed Systems (OPODIS'2006). (2006) 81–93 Also published as a Brief

Announcement in the Proceedings of the 20th International Symposium on Distributed Computing (DISC'2006).
26. Schiper, N., Pedone, F.: Solving atomic multicast when groups crash. In: OPODIS '08: Proceedings of the 12th International Conference on Principles of Distributed Systems, Berlin, Heidelberg, Springer-Verlag (2008) 481–495
27. Guerraoui, R., Schiper, A.: Genuine atomic multicast in asynchronous distributed systems. Theoretical Computer Science (Elsevier) **254** (2001) 297–316
28. Schiper, N., Sutra, P., Pedone, F.: P-Store: Genuine Partial Replication in Wide Area Networks. Technical Report USI-INF-TR-2010-3, Faculty of Informatics, Università della Svizzera italiana, Lugano, Switzerland (April 2010)
29. Ye, H., Kerherve, B., v Bochmann, G., Bourne, D.: Towards database scalability through efficient data distribution in e-commerce environments. In: Electronic Commerce, 2002. Proceedings. Third International Symposium on. (2002) 87 – 95
30. Romano, P., Carvalho, N., Rodrigues, L.: Towards distributed software transactional memory systems. In: Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware. LADIS '08, New York, NY, USA, ACM (2008) 4:1–4:4
31. Kotselidis, C., Ansari, M., Jarvis, K., Luján, M., Kirkham, C., Watson, I.: Distm: A software transactional memory framework for clusters. In: In Proc. of the International Conference on Parallel Processing (ICPP). (2008) 51–58
32. Hammond, L., Wong, V., Chen, M., Carlstrom, B.D., Davis, J.D., Hertzberg, B., Prabhu, M.K., Wijaya, H., Kozyrakis, C., Olukotun, K.: Transactional memory coherence and consistency. In: Proceedings of the 31st annual international symposium on Computer architecture. ISCA '04, Washington, DC, USA, IEEE Computer Society (2004) 102–
33. Gray, C., Cheriton, D.: Leases: an efficient fault-tolerant mechanism for distributed file cache consistency. In: Proceedings of the twelfth ACM symposium on Operating systems principles. SOSP '89, New York, NY, USA, ACM (1989) 202–210
34. Osrael, J.: Replication Techniques for Balancing Data Integrity with Availability. PhD thesis, Institut fur Softwaretechnik und Interaktive Systeme, Arbeitsbereich fur Business Informaticsu Technische Universitat Wien (2007)
35. Zou, H., Jahanian, F.: Optimization of a real-time primary-backup replication service. Technical report, Parallel and Distributed Systems, IEEE Transactions on (1998)
36. Schneider, F.: The state machine approach: A tutorial. In Simons, B., Spector, A., eds.: Fault-Tolerant Distributed Computing. Volume 448 of Lecture Notes in Computer Science. Springer Berlin - Heidelberg (1990) 18–41
37. Atif, M.: Analysis and verification of two-phase commit and three-phase commit protocols. In: Emerging Technologies, 2009. ICET 2009. International Conference on. (2009) 326 –331
38. Pedone, F., Guerraoui, R., Schiper, A.: Exploiting Atomic Broadcast in Replicated Databases. Technical report (1998)
39. Pedone, F., Guerraoui, R., Schiper, A.: The Database State Machine Approach. Technical report (1999)
40. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data, New York, NY, USA, PGS98b/LSRACM (1996) 173–182
41. Couceiro, M., Romano, P., Carvalho, N., Rodrigues, L.: D2STM: Dependable Distributed Software Transactional Memory. In: Proceedings of the 2009 15th

IEEE Pacific Rim International Symposium on Dependable Computing. PRDC '09, Washington, DC, USA, IEEE Computer Society (Nov 2009) 307–313
42. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7) (1970) 422–426
43. Cachopo, J.a., Rito-Silva, A.: Versioned boxes as the basis for memory transactions. Sci. Comput. Program. **63** (December 2006) 172–185
44. 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)
45. Pedone, F., Schiper, A.: Optimistic atomic broadcast. In Kutten, S., ed.: Distributed Computing. Volume 1499 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1998) 318–332 10.1007/BFb0056492.
46. Coulon, C., Pacitti, E., Valduriez, P.: Consistency management for partial replication in a high performance database cluster. In: Parallel and Distributed Systems, 2005. Proceedings. 11th International Conference on. Volume 1. (2005) 809 – 815 Vol. 1
47. Alonso, G.: Partial database replication and group communication primitives (extended abstract). In: in Proceedings of the 2 nd European Research Seminar on Advances in Distributed Systems (ERSADS'97. (1997) 171–176
48. Sousa, A., Pedone, F., Moura, F., Oliveira, R.: Partial replication in the database state machine. In: n Proc. of the IEEE International Symposium on Network Computing and Applications (NCA 2001), IEEE CS (October 2001) 298–309
49. 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: Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing, Washington, DC, USA, IEEE Computer Society (2007) 290–297
50. Vicente, P., Rodrigues, L.: An indulgent uniform total order algorithm with optimistic delivery. In: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems. SRDS '02, Washington, DC, USA, IEEE Computer Society (2002) 92–
51. U. Fritzk, J., Ingels, P., Mostéfaoui, A., Raynal, M.: Fault-tolerant total order multicast to asynchronous groups. In: Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems. SRDS '98, Washington, DC, USA, IEEE Computer Society (1998) 228–