# Uma Linguagem de Domínio Específico para Transações Distribuídas em Java

JERÔNIMO DA CUNHA RAMOS[1]; MAURÍCIO LIMA PILLA[1]; ANDRÉ RAUBER DU BOIS[1]

[1]*Universidade Federal de Pelotas – {jdcramos, pilla, dubois}@inf.ufpel.edu.br*

## 1. INTRODUÇÃO

Para tirar proveito do paralelismo disponibilizado pelos computadores atuais, os programas devem conter atividades que possam executar concorrentemente. Normalmente, em arquiteturas de memória compartilhada, isto é feito com o uso de *threads*, variáveis compartilhadas e *locks*. Porém, este modelo de programação impõe quase toda a complexidade de exploração do paralelismo e de tratamento das condições de corrida ao programador, o que torna-o propenso a erros, como *deadlocks*. A comunidade científica vem propondo alternativas a este modelo para facilitar e popularizar a programação concorrente, como é o caso das memórias transacionais (RIGO et al., 2007).

Memória transacional é uma abstração para programação concorrente baseada na ideia de transações, parecidas com as de banco de dados. Uma transação de memória é uma sequência atômica de operações que modificam a memória e que podem ser executadas completamente ou podem ser abortadas. Desta maneira, transações de memória executam como se estivessem modificando uma área de memória isolada do acesso de outras transações. Estas só conseguem ver o resultado após o *commit (DU BOIS, 2008)*.

Existem muitas vantagens no uso deste modelo ao invés dos convencionais (mutex, semáforos, monitores, etc). Uma delas é a facilidade de programação, pois basta que o programador utilize um bloco atômico para marcar os trechos onde podem ocorrer condições de corrida e o sistema transacional se encarrega de garantir a atomicidade da execução do bloco. Outras vantagens são ausência de deadlock, composabilidade.

Sendo assim, o modelo de memórias transacionais tem se mostrado promissor para máquinas SMP (*Symmetric Multi Processor*), no entanto existem poucos trabalhos na área focados em arquiteturas distribuídas. O estudo destas arquiteturas continua extremamente importante nos dias de hoje, tanto por comporem uma grande parcela da computação de alto desempenho, quanto pela tendência de cada vez mais dispositivos computacionais se comunicarem entre si.

Nos últimos anos, as pesquisas sobre Memórias Transacionais têm sido bastante focadas em máquinas *multicore*, deixando outros tipos de arquiteturas, como *clusters* e máquinas NUMA (*Non-Uniform Memory Access*) quase inexploradas (LU et al., 2010). Estas arquiteturas tem como principal diferencial o tempo variado de acesso à memória local e remota. No caso dos *clusters* e outras arquiteturas distribuídas, cada nodo possui seu próprio espaço de endereçamento de memória, fazendo com que a comunicação tenha que ser feita por troca de mensagens e, geralmente, não haja coerência de cache. Além disto, outra dificuldade clássica das arquiteturas distribuídas é a não existência de um relógio global para a ordenação dos eventos, o que pode ser resolvido com a aplicação do algoritmo de Lamport (LAMPORT, 1978).

Existem diversas implementações de sistemas de memórias transacionais para máquinas *multicore*, tais como TinySTM (FELBER et al., 2008), SwissTM (DRAGOJEVIĆ et al., 2009), AdaptSTM (PAYER; GROSS, 2011) e CMTJava (DU

BOIS; ECHEVARRIA, 2009), porém o foco destas não está em arquiteturas distribuídas, sendo assim não preveem soluções para os dificultantes citados acima. As propostas focadas em arquiteturas distribuídas são pouco numerosas, sendo estas focadas em *clusters*, como é o caso de Cluster-STM (BOCCHINO et al., 2008), DiSTM (KOTSELIDIS, 2008) e TFA (SAAD; RAVINDRAN, 2012).

O modelo de memórias transacionais tem se mostrado bastante promissor e largamente estudado para aplicação em máquinas *multicore*. Acredita-se que este modelo também possa ser utilizado, com algumas adaptações, para facilitar a programação de arquiteturas distribuídas, como os *clusters*. Sendo assim, o objetivo principal deste trabalho é propor a linguagem DCMTJava, uma linguagem de domínio específico embarcada baseada em mônadas, que contempla tanto transações locais quanto transações envolvendo objetos remotos.

## 2. METODOLOGIA

Foi realizada uma revisão bibliográfica para compreender os conceitos de memórias transacionais e arquiteturas distribuídas. O principal objetivo deste estudo foi identificar as características das implementações de cada modelo já existente e os principais desafios para estender algum destes modelos de forma que disponibilize suporte à transações distribuídas. Optou-se por estender a linguagem CMTJava e utilizar o algoritmo TFA para lidar com os objetos distribuídos. Após esta primeira etapa, deu-se início às implementações.

CMTJava é uma linguagem de domínio especifico que estende a linguagem Java, adicionando suporte para programação de STM. Ela oferece a abstração de objetos transacionais. Em CMTJava, para definir um desses objetos basta estender a interface `TObject` e os atributos destes objetos poderão ser acessados somente através de métodos especiais de `get` e `set`, gerados automaticamente pelo compilador. Estes métodos retornam ações transacionais como resultado e estas somente podem ser executadas dentro de blocos atômicos. Com isto, garante-se as propriedades de atomicidade e isolamento. Algumas abstrações de alto nível da CMTJava, como os objetos transacionais e os blocos de ações transacionais (STMDO) são baseadas em uma linguagem de mais baixo nível composta de mônadas, *closures* e chamadas à bibliotecas que implementam transações. Logo, para a execução dos programas escritos na linguagem é necessário transformar essas abstrações de alto nível para as construções de baixo nível disponíveis, como pode ser visto na Tabela 1.

| **CMTJava** | **Java + *Closures*** |
|---|---|
| STM{type var <- e; s} | STMRTS.bind (e, {type var -> STM {s}}) |
| STM{e; s} | STMRTS.them (e, STM{s}) |
| STM{e} | e |

**Tabela 1:** Esquema de tradução dos blocos STMDO

A implementação atual da CMTJava (BANDEIRA, 2013) contempla apenas transações locais. Propõe-se aqui a linguagem DCMTJava, que estenderá a CMTJava de forma a também dar suporte a transações envolvendo objetos distribuídos. A primeira extensão será baseada no algoritmo TFA, proposto em(SAAD; RAVINDRAN, 2012). Este algoritmo trabalha a nível de objetos e possui o diferencial de mover os objetos através dos nodos, ao invés de mover instruções de controle. Uma comparação entre TFA e outros dois trabalhos focados em memória transacional distribuída pode ser vista em (RAMOS, 2013).

Após a extensão, TObjects poderão ser movidos de um nodo à outro. Suas classes serão marcadas com implements RemoteTObject para que o compilador as reconheça e gere seus métodos especiais. Estes tem como objetivo facilitar a serialização, envio e checagem de conflitos entre os nodos. Dessa forma, a mônada STM deve ser estendida para lidar com as operações distribuídas. Além destes métodos dos RemoteTObjects, será necessária uma interface para algum tipo de diretório, para a localização e envio dos objetos.

## 3. RESULTADOS E DISCUSSÃO

A fase de revisão bibliográfica já foi concluída e o trabalho encontra-se em fase de implementações iniciais. Dos resultados práticos, vale ressaltar a possibilidade de mover os TObjects entre máquinas e o registro e localização destes, sendo estes requisitos básicos para a implementação do sistema transacional distribuído.

Mover os TObjects foi um dificultante inesperado. Objetos comuns da linguagem Java podem ser serializados facilmente se implementarem a interface Serializable, porém, o mesmo não ocorre com os TObjects. O motivo disto é o uso de *Closures*, que não são serializáveis. Para contornar este problema, foi criada uma classe somente para representação dos dados dos TObjetcs e esta é serializável. Sendo assim, para enviar um TObject, ele é convertido em um objeto desta classe de representação e reconvertido em TObject pelo nodo receptor.

O registro e localização dos objetos está sendo realizada de maneira simples, com um servidor centralizado que contem a tabela de localização dos objetos e recebe as requisições via *socket*. Sabe-se que ele pode se tornar um gargalo se o número de nodos for grande, porém optou-se por esta estratégia para acelerar a prototipação.

O próximo passo de implementação é a alteração do sistema transacional para que utilize o algoritmo TFA. Concluída esta etapa, será possível realizar experimentos para verificar o desempenho do modelo proposto.

## 4. CONCLUSÕES

Este trabalho propõe uma linguagem de programação de domínio específico embarcada para transações distribuídas, utilizando mônadas para representar as transações. O modelo de programação com memória transacional tem se mostrado bastante promissor, com diversas vantagens sobre os modelos clássicos de sincronização, porém as arquiteturas distribuídas apresentam características que não são levadas em consideração pela maior parte das implementações de STM, tais como localização dos objetos e características da rede, sendo estes os enfoques principais deste estudo.

Alguns trabalhos futuros previstos são a conclusão das implementações, para que possam ser realizados testes e a validação do modelo. Também seria interessante a implementação de outras estratégias para lidar com objetos que não podem ser movidos, que poderiam vir a integrar a linguagem, formando uma estratégia mista.

## 5. REFERÊNCIAS BIBLIOGRÁFICAS

BANDEIRA, R. L. **Um Sistema de Detecção de Conflitos com Invalidação Mista para a Linguagem CMTJava.** 2013. Dissertação (Mestrado em Computação) – Programa de Pós-graduação em Computação, UFPel.

BOCCHINO, R. L.; ADVE, V. S.; CHAMBERLAIN, B. L. Software Transactional Memory for Large Scale Clusters. **Proceedings of the 13th ACM SIGPLAN Symposium on PPoPP,** New York, p.247-258, 2008.

DRAGOJEVIĆ, A; GUERRAOUI, R; KAPALKA, M. Stretching Transactional Memory. **Proceedings of the 2009 ACM SIGPLAN Conference on PLDI,** New York, p.155-165, 2009.

DU BOIS, A. R. Memórias Transacionais e Troca de Mensagens: Duas Alternativas para a Programação de Máquinas Multi-Core. **SBC, P. A. (Ed.). Escola Regional de Alto Desempenho,** p.43-76, 2008.

DU BOIS, A. R.; ECHEVARRIA, M. A. Domain Specific Language for Composable Memory Transactions in Java. **DSL '09: Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages,** p.170-186, 2009.

FELBER, P.; FETZER, C; RIEGEL, T. Dynamic Performance Tuning of Word-Based Software Transactional Memory. **Proceedings of the 13th ACM SIGPLAN Symposium on PPoPP,** New York, p.237-246, 2008.

KOTSELIDIS, C.; ANSARI, M.; JARVIS, K.; LUJÁN, M.; KIRKHAM, C.; WATSON, I. DiSTM: A Software Transactional Memory Framework for Clusters. **Proceedings of 37th International Conference on Parallel Processing,** Washington, p.51-58, 2008.

LAMPORT, L. Time, Clocks, and the Ordering of Events in a Distributed System. **Commun. ACM,** v.21, n.7, p.558-565, 1978.

Lu, K.; Wang, R.; Lu, X. Brief Announcement: NUMA-Aware Transactional Memory. **Proceedings of the 29th ACM SIGACT-SIGOPS,** New York, p.69-70, 2010.

PAYER, M.; GROSS, T. R.; Performance Evaluation of Adaptivity in Software Transactional Memory. **Performance Analysis of Systems and Software (ISPASS), 2011 IEEE International Symposium,** p.165-174, 2011.

RAMOS, J. C. **Um Estudo Sobre Memórias Transacionais para Arquiteturas Distribuídas.** 2013. Trabalho Individual (Mestrado em Computação) – Programa de Pós-graduação em Computação, UFPel.

RIGO, S.; CENTODUCATTE, P.; BALDASSIN, A. Memórias Transacionais: Uma Nova Alternativa para Programação Concorrente. **Minicursos do VIII WSCAD,** 2007.

SAAD, M. M.; RAVINDRAN, B. Transactional Forwarding: Supporting Highly-Concurrent STM in Asynchronous Distributed Systems. **Computer Architecture and High Performance Computing (SBAC-PAD), 2012 IEEE 24th International Symposium,** p.219-226 , 2012.