

**Michael Alexandre Costa**

**Memórias Transacionais**

Trabalho Individual apresentado ao Programa de Pós-Graduação em Computação da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Mestre em Ciência da Computação

Orientador: Prof. Dr. André Rauber Du Bois  
Coorientador: Prof. Dr. Mauricio Lima Pilla

Pelotas, 2018

## RESUMO

COSTA, Michael Alexandre. **Memórias Transacionais**. 2018. 20 f. Trabalho Individual (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2018.

...

**Palavras-Chave:** memória transacional; numa; uma; escalonamento

## ABSTRACT

COSTA, Michael Alexandre. **Transaccional Memory**. 2018. 20 f. Trabalho Individual (Mestrado em Ciência da Computação) – Programa de Pós-Graduação em Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2018.

...

**Keywords:** transaccional memory; numa; uma; scheduler

## LISTA DE FIGURAS

Figura 1	Exemplo de versionamento adiantado (a) e atrasado (b). Fonte: (BALDASSIN, 2009) . . . . .	9
Figura 2	Detecção de conflitos em modo adiantado. Fonte: (RIGO; CENTO-DUCATTE; BALDASSIN, 2007) . . . . .	10
Figura 3	Detecção de conflitos em modo atrasado. Fonte: (RIGO; CENTO-DUCATTE; BALDASSIN, 2007) . . . . .	11
Figura 4	Classificação das técnicas de escalonamento. Fonte: (SANZO, 2017)	13

## **LISTA DE TABELAS**

## **LISTA DE ABREVIATURAS E SIGLAS**

STM	Software Transactional Memory
TM	Transactional Memory
NUMA	Non-Uniform Memory Access

# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	<b>7</b>
1.1	Uma subseção	7
<b>2</b>	<b>MEMÓRIA TRANSACIONAL</b>	<b>8</b>
2.1	Propriedades	8
2.2	Versionamento de Dados	9
2.3	Detecção de Conflito	9
<b>3</b>	<b>ESCALONAMENTO DE STM</b>	<b>12</b>
3.1	Características e Técnicas	12
3.1.1	Técnica baseada em Feedback	13
3.1.2	Técnica baseada em Previsão	13
3.1.3	Técnicas Reativas	13
3.1.4	Técnica baseada em Heurística Mista	13
3.1.5	Machine Learning-Based Techniques	14
3.1.6	Analytical Model-Based Techniques	14
3.1.7	Mixed Model-Based Techniques	14
<b>4</b>	<b>ESCALONADORES NUMA</b>	<b>15</b>
<b>5</b>	<b>ESCALONAMENTO DE TRANSAÇÕES APLICADO À NUMA</b>	<b>16</b>
<b>6</b>	<b>DISCUSSÕES</b>	<b>17</b>
<b>7</b>	<b>CONCLUSÃO</b>	<b>18</b>
	REFERÊNCIAS	19
	ANEXO A UM ANEXO	20

# **1 INTRODUÇÃO**

...

## **1.1 Uma subseção**

...



## 2 MEMÓRIA TRANSACIONAL

Memória Transacional, ou *Transactional Memory* (TM), é uma classe de mecanismos de sincronização que fornece uma execução atômica e isolada de alterações em um conjunto de dados compartilhados. Estas estão sendo desenvolvidas para que no futuro tornem-se o principal meio de fazer a sincronização em um programa concorrente, substituindo a sincronização baseada em *locks* (MORESHET; BAHAR; HERLIHY, 2006). As TMs podem ser implementadas em *software* (STM), em *hardware* (HTM) ou ainda em uma versão híbrida de *hardware* e *software*.

Na programação utilizando STMs, todo o acesso à memória compartilhada é realizado dentro de transações e todas as transações são executadas atomicamente em relação a transações concorrentes.

A principal vantagem na programação usando STM é que o programador apenas delimita as seções críticas e não é necessário preocupar-se com a aquisição e liberação de *locks*. Os *locks*, quando utilizados de forma incorreta, podem levar a problemas como *deadlocks* (BANDEIRA, 2010).

### 2.1 Propriedades

Transação é uma sequência finita de escritas e leituras na memória executada por uma *thread* (HERLIHY; ELIOT; MOSS, 1993), e deve satisfazer três propriedades:

- **Atomicidade:** cada transação faz uma sequência de mudanças provisórias na memória compartilhada. Quando a transação é concluída, pode ocorrer um *commit*, tornando suas mudanças visíveis a outras *threads* instantaneamente, ou pode ocorrer um *abort*, fazendo com que suas alterações sejam descartadas;
- **Consistência:** as transações devem garantir que um sistema consistente deve ser mantido consistente. Esta propriedade está relacionada com o conceito de invariância;
- **Isolamento:** as transações não interferem nas execuções de outras transações, assim parecendo que elas são executadas serialmente. Uma transação não

observa o estado intermediário de outra.

## 2.2 Versionamento de Dados

O versionamento de dados faz é responsável pelo gerenciamento das versões dos dados. Ele armazena tanto o valor do dado no início de uma transação como também o valor do dado modificado durante a transação, isso para garantir a propriedade de atomicidade (BALDASSIN, 2009).

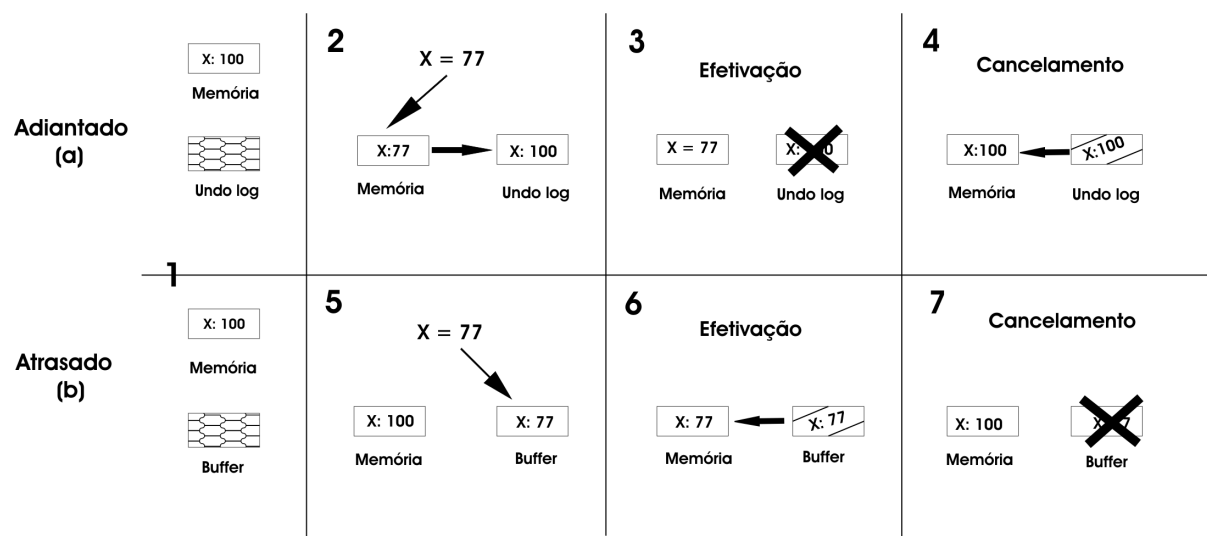


Figura 1 – Exemplo de versionamento adiantado (a) e atrasado (b). Fonte: (BALDASSIN, 2009)

Existem dois tipos de versionamento de dados:

- **Versionamento Adiantado:** como pode ser visto na Figura 1 (a), o valor modificado durante a transação é armazenado direto na memória e o valor inicial é armazenado em um *undo log*, para que no caso de cancelamento na transação o valor inicial seja restaurado na memória.
- **Versionamento Atrasado:** como pode ser visto na Figura 1 (b) neste versionamento o valor modificado durante a transação é armazenado em um *buffer* e o valor inicial é mantido na memória até que aconteça um *commit* na transação, onde o valor armazenado no *buffer* é escrito na memória. Caso aconteça o cancelamento na transação, o valor do *buffer* é descartado.

## 2.3 Detecção de Conflito

Mecanismos de detecção de conflitos verificam a existência de operações conflitantes durante uma transação. Um conflito ocorre quando duas transações estão

acessando um mesmo dado na memória e pelo menos uma das transações está fazendo uma operação de escrita (BALDASSIN, 2009).

Da mesma forma que o versionamento de dados, a detecção de conflito também pode ser de dois tipos:

- **Detecção de Conflitos Adiantado:** ocorrem no momento em que duas transações acessam um mesmo dado e uma delas faz uma operação de escrita. Essa operação de escrita é detectada e então uma transação é abortada. Neste tipo de detecção pode ocorrer o problema chamado de *livelock*, quando duas transações ficam cancelando-se, desta forma, a execução do programa não progride. A Figura 2 mostra como é feita a detecção de conflitos adiantado.

O Caso 1, mostra a execução sem conflitos, onde as duas transações são executadas sem problemas. Já o Caso 2, mostra o que acontece quando ocorre um conflito, onde T1 lê A e logo depois T2 escreve em A, então o conflito é detectado e T1 é abortada, após ser efetivada T2, a transação T1 consegue ler A sem problema de conflito. Por fim o Caso 3 mostra a situação de *livelock*, onde as duas transações tentam ler e escrever em A, assim as duas acabam sempre se abortando.

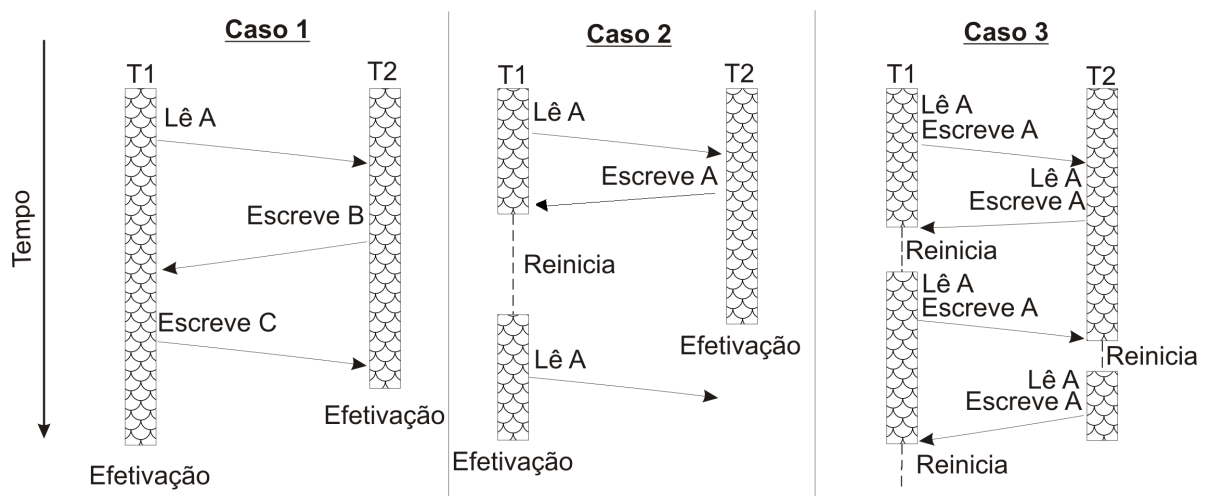


Figura 2 – Detecção de conflitos em modo adiantado. Fonte: (RIGO; CENTODUCCATTE; BALDASSIN, 2007)

- **Detecção de Conflitos Atrasado:** Este tipo de detecção de conflito ocorre no final da transação. Antes da transação ser efetuada, é verificado se ocorreu um conflito. Caso tenha ocorrido, a transação é cancelada, senão é efetivada. Para transações muito grandes não é recomendado este tipo de detecção, pois uma transação grande pode ser abortada várias vezes por transações pequenas, assim gastando tempo de processamento desnecessário, este problema se chama *starvation*. A Figura 3 mostra como é feita a detecção de conflitos atrasado.

O Caso 1, mostra as transações acessando dados diferentes, não ocasionando conflitos. No Caso 2, T2 lê A que é escrita por T1. A T2 só nota o conflito quando T1 é efetivado. Logo depois de notar o conflito T2 é abortada. No Caso 3 não ocorre nenhum conflito, pois T1 lê A antes de T2 escrever. O Caso 4 mostra a situação em que, após ser cancelada, T1 volta a executar.

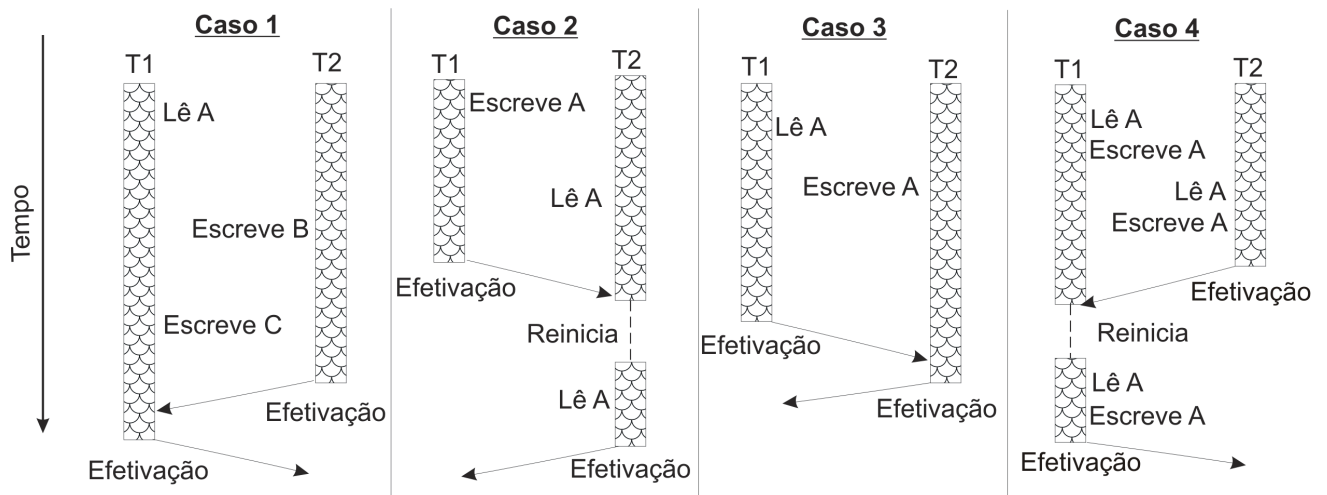


Figura 3 – Detecção de conflitos em modo atrasado. Fonte: (RIGO; CENTODUCATTE; BALDASSIN, 2007)

Para solucionar o problema de qual transação continuará executando, quando ocorre um conflito, é utilizado um gerenciador de contenção (HARRIS; LARUS; RAJWAR, 2010). O gerenciador de contenção é o responsável por decidir quando e qual transação vai ser abortada, isso para garantir que a execução do programa prossiga sem problemas.

### 3 ESCALONAMENTO DE STM

A bibliografia de STM apresenta um gama distinta e extensa de estratégias de escalonamento. Estas diferem-se devido as distintas características de aplicações encontradas. Nas quais podem apresentar maior ou menor nível de contenção, diferentes read-sets e write-sets, entre outras características.

Os algoritmos de escalonamento visam otimizar o tempo de execução. Para isto devem inserir o menor *overhead* possível no código propondo uma estratégia eficiente para as características do problema abordado. Alguns algoritmos estudados como *To do* (??) busca serializar transações conflitantes com a utilização de filas.

Os escalonadores de STM estudados na bibliografia caracterizam-se por executar o escalonamento em dois níveis distintos. Os dois níveis buscam reduzir o tempo de execução e garantir a execução do programa. Estes escalonamentos de STM são o escalonamento de *threads* e de transações.

O escalonamento de *threads*, busca executar sua tomada de decisão sobre as *threads* ativas no programa, podendo adicionar mais *threads*, remove-las, ou migrá-las entre os *cores*.

O escalonamento de transações, busca executar sua tomada de decisão sobre as transações do programa, estas podem estar em execução, já terem sido abortadas, ou serem a próxima a executar.

As tomadas de decisões dos escalonadores citados acima se dá a partir de outras características. Estas avaliam de forma diferente os dados para a tomada de decisão e em tempos de execução diferentes, assim como, podem efetuar operações distintas conforme suas premissas de escalonamento.

#### 3.1 Características e Técnicas

O trabalho (SANZO, 2017), classifica as técnicas de escalonamento de STM como, baseadas em heurísticas e baseadas em modelos, esta classificação é apresentada na Figura 4, onde cada classe, apresenta técnicas distintas de escalonamento.

Estas diferentes técnicas apresentadas foram exploradas e exemplificadas no tra-

balho citado acima. As subseções a seguir vão abordar e detalhar estas técnicas, assim como seus algoritmos.

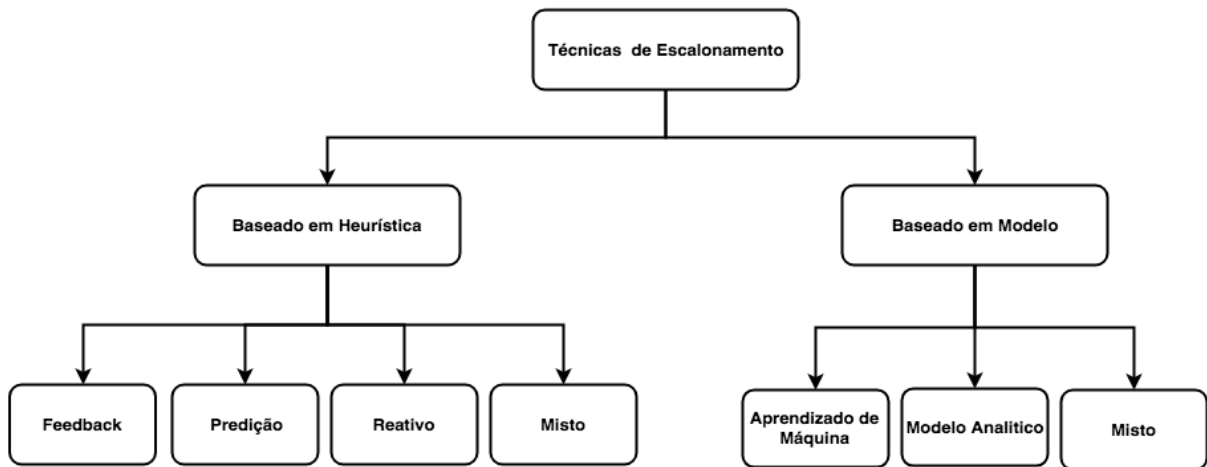


Figura 4 – Classificação das técnicas de escalonamento. Fonte: (SANZO, 2017)

### 3.1.1 Técnica baseada em Feedback

As técnicas baseadas em feedback utilizam um sistema de loop fechado para controle de um sistema dinâmico. Essa técnica utiliza um parâmetro de desempenho que é avaliado pelo escalonador a cada iteração do loop para decidir a ação a ser executada.

### 3.1.2 Técnica baseada em Previsão

As técnicas baseadas em previsão utilizam probabilidade para tomar a decisão apropriada para sua execução. Essa técnica utiliza um conjunto de dados de um mesmo parâmetro para estimar a probabilidade de execução, assim o escalonador toma a decisão de execução dinamicamente.

### 3.1.3 Técnicas Reativas

As técnicas reativas adotam uma estratégia onde o escalonador entra em ação após uma transação ser abortada, isso busca evitar conflitos repetidos. Esta estratégia baseia-se na suposição de que os dados acessados por uma transação abortada tem a possibilidade de serem os mesmos acessados durante sua reexecução, ocasionando novo conflito com a transação ainda em execução.

### 3.1.4 Técnica baseada em Heurística Mista

As técnicas baseadas em heurísticas mista buscam mesclar as diferentes técnicas comentadas acima, para tirar proveito de diferentes perfis de transações. Assim, o escalonador pode tomar a melhor decisão de execução de acordo com o perfil de

execução das transações.

### **3.1.5 Machine Learning-Based Techniques**

...

### **3.1.6 Analytical Model-Based Techniques**

...

### **3.1.7 Mixed Model-Based Techniques**

...

## **4 ESCALONADORES NUMA**

...



## **5 ESCALONAMENTO DE TRANSAÇÕES APLICADO À NUMA**

...

## **6 DISCUSSÕES**

...

## **7 CONCLUSÃO**

...

## REFERÊNCIAS

BALDASSIN, A. J. **Explorando Memória Transacional em Software nos Contextos de Arquiteturas Assimétricas, Jogos Computacionais e Consumo de Energia**. 2009. Dissertação de Doutorado — Universidade Estadual de Campinas.

BANDEIRA, R. de Leão. **Compilador para a linguagem CMTJava**. 2010. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) — Universidade Federal de Pelotas.

HARRIS, T.; LARUS, J.; RAJWAR, R. Transactional Memory, 2nd edition. **Synthesis Lectures on Computer Architecture**, [S.l.], v.5, n.1, p.1–263, 2010.

HERLIHY, M.; ELIOT, J.; MOSS, B. Transactional Memory: Architectural Support for Lock-Free Data Structures. In: PROC. OF THE 20TH ANNUAL INTL. SYMPOSIUM ON COMPUTER ARCHITECTURE, 1993. **Anais...** [S.l.: s.n.], 1993. p.289–300.

MORESHET, T.; BAHAR, R. I.; HERLIHY, M. Energy-Aware Microprocessor Synchronization: Transactional Memory vs. Locks. In: WORKSHOP ON MEMORY PERFORMANCE ISSUES, 2006. **Proceedings...** [S.l.: s.n.], 2006.

RIGO, S.; CENTODUCATTE, P.; BALDASSIN, A. **Memórias Transacionais: Uma Nova Alternativa para Programação Concorrente**. [S.l.]: In Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing, 2007.

SANZO, P. D. Analysis, Classification and Comparison of Scheduling Techniques for Software Transactional Memories. **IEEE Transactions on Parallel and Distributed Systems**, [S.l.], v.28, n.12, p.3356–3373, Dec 2017.

## **ANEXO A   UM ANEXO**

...