Analysis of Polka Contention Manager for use in Multicore Hard Real-Time Systems

Adrien Quillet, Audrey Queudet, and Didier Lime

LS2N, University of Nantes, France

June 10^{th}, 2020

Plan

Introduction

Introduction

- *Shared resources management* in multicore hard real-time systems
- *Software Transactional Memory*-based (*STM*-based) approach
- *Conflict resolution* resolved by consulting an STM-level contention manager.
- Timing analysis of *Polka* contention manager

Problematic

Can STMs using Polka for resolving conflicts between competing transactions be used in multicore hard real-time systems ?

Introduction
Why Transactional Memory ?

Locks are difficult to manage

- deadlock
- starvation
- priority inversion
- lock convoy
- reduced parallelism
- complexity of multi-thread programming

Transactional memory offers an interesting alternative.

Plan

Transactional Memory Concepts

Advantages of transactional memory

Transactional memory

- Easy to use synchronization construct
- Failure atomicity & recovery
- Composability

Memory transaction

- An atomic & isolated sequence of memory accesses
- Inspired by database transactions

ACID properties [1]

- *Atomicity*
- *Consistency*
- *Isolation*
- *Durability*

1. Theo Haerder and Andreas Reuter. *Principles of transaction-oriented database recovery*. ACM Computing Surveys (CSUR), 15(4) :287–317, 1983

Transactional Memory Concepts

TM implementation basics

Types of implementations

- Hardware transactional memory (HTM)
- Software transactional memory (STM)
- Hybrid transactional memory (Hybrid TM)

Transactional Memory Concepts

TM and Real-time systems

Progress guarantees

- *obstruction-free*
- *lock-free*
- *wait-free*

TM and Hard Real-time systems

- Only wait-free TMs *"guarantee that any process can complete any operation in a finite number of steps, regardless of the execution speeds on the other processes"*[1]
- The lack of upper bounds on the execution time of transactions prevents the use of TM in real-time systems.

1. M. Herlihy. *Wait-free synchronization*. ACM Transactions on Programming Languages and Systems, 13 :1 :124–149, 1991

Software Transactional Memory Concepts

Basic implementation requirements

- Data versioning
- Conflict detection
  - eager
  - lazy
- Conflict resolution
  - contention management

Concurrency control

An STM needs *a contention manager* module which is in charge of resolving conflicts between transactions.

Plan

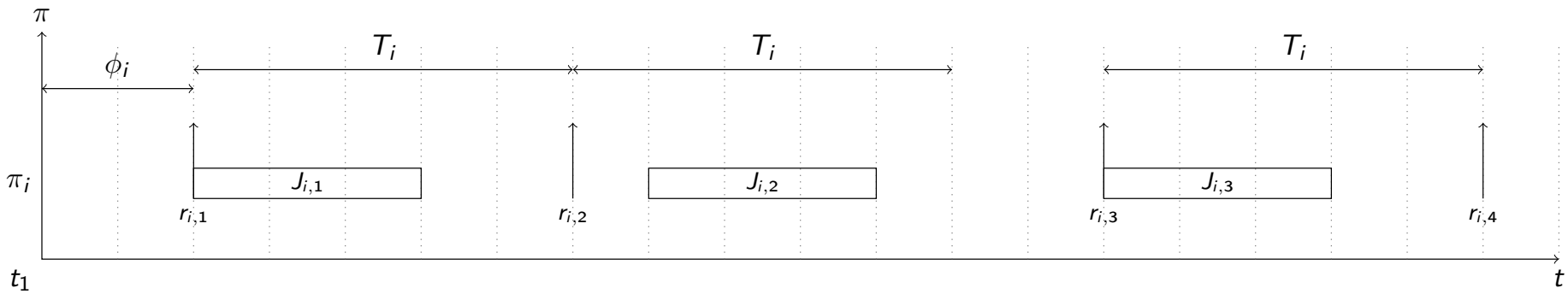
System model
Platform, Objects, and Operations

We consider a platform made of m identical processor cores $\pi = \{\pi_1, \ldots, \pi_m\}$.
Every shared resource managed by an STM is actually manipulated with *an object* o.
We consider three possible system *operations* :

- $read(o)$
- $write(o, x)$
- $nop()$

Task model

A task τ_i is a tuple $\langle \pi_i, \phi_i, T_i, \Sigma_i \rangle$.
Each task τ_i produces an infinite number of successive jobs $J_{i,j}$, with $j \in \mathbb{N}^*$.

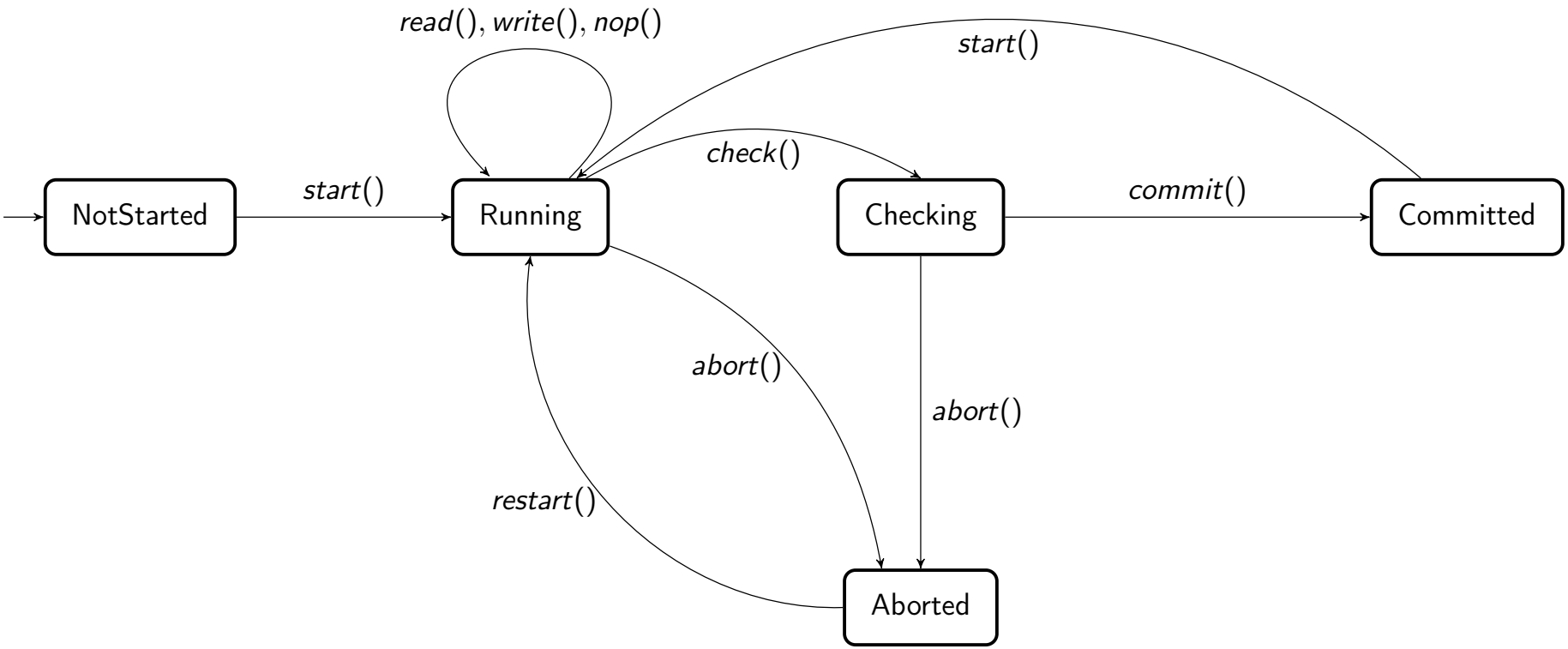


An active job can be preempted at some time t_n, except if it is executing a transaction.

Transaction model

State-transition diagram of possible transaction status

A transaction σ_i is a tuple $\langle b_i, N_i, \Omega_i, O_i \rangle$.



Transaction model
Properties

Definition 5.4

A transaction is said *timely correct* if the maximum period between its (*start()*|*restart()*) and (*abort()* | *commit()*) operations is less than or equal to TT. Otherwise (i.e. this period is exceeded), the transaction is said *timely incorrect*.

TT

denotes the upper bound on the time during which transactions can remain *active* (i.e. its status is either *Running* or *Checking*) in the system.

Plan

Polka Contention Manager

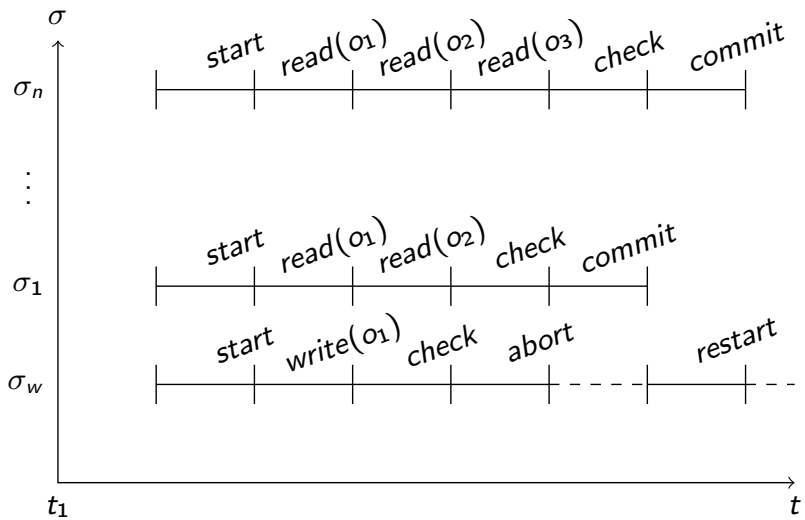
Polka contention policy [1]

- the priority of a transaction (called *karma*) is increased by one whenever the transaction acquires a shared resource
- transactions start with a zero *karma*
- when a transaction commits, its *karma* returns to zero
- when a transaction aborts, its *karma* is incremented by one
- in case of conflicts, the checking transaction makes a number of attempts equal to the difference among priorities of the transactions before aborting the competing transaction
- exponential backoff between attempts

1. W.-N. Scherer III and Michael L Scott. *Advanced contention management for dynamic software transactional memory*, Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing. ACM, pp. 240–248, 2005

Polka Contention Manager

n "slow" readers and one "fast" writer



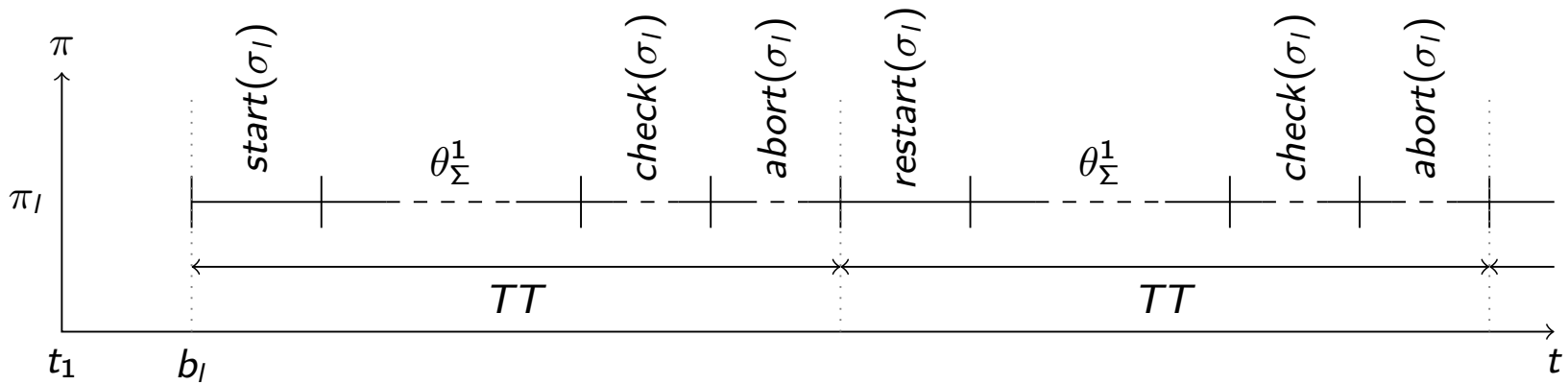
The writer's *karma* is lower than the *karma* of readers, the writer will sleep, waiting for readers to commit.

Slowest Karma Accumulation

Lemma 5.1

In a system using a software transactional memory whose contention manager is Polka, the slowest karma accumulation at time t_n of a timely correct transaction σ_l is given by :

$$k_l = \left\lfloor \frac{t_n - b_l}{TT} \right\rfloor \times 2 + \epsilon_l \text{ with } \epsilon_l \in \{0, 1\}$$

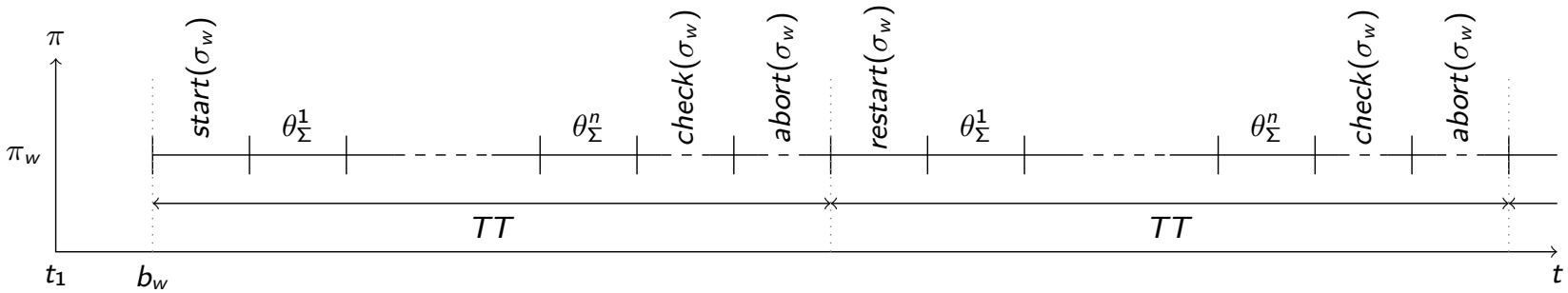


Fastest Karma Accumulation

Lemma 5.2

In a system using an STM whose contention manager is Polka, whose conflict detection is lazy and in which all transactions are timely correct, the fastest karma accumulation at time t_n of transaction σ_w is given by :

$$k_w = \left\lfloor \frac{t_n - b_w}{TT} \right\rfloor \times (TT - 2) + \epsilon_w$$
$$\text{with } \epsilon_w \in \{0, \ldots, n\}$$



Achieving the Greatest Karma

Lemma 5.3

In a system using a software transactional memory whose contention manager is Polka, whose conflict detection is *lazy* and in which all transactions are timely correct, the maximum possible karma value is

$$k_S^{max} = \Big(min\left(|\pi|, |\Sigma|\right) - 1\Big) \times (TT - 2) + \epsilon$$

$$\text{with } \epsilon \in \{0, \ldots, n\}$$

Lemma 5.4

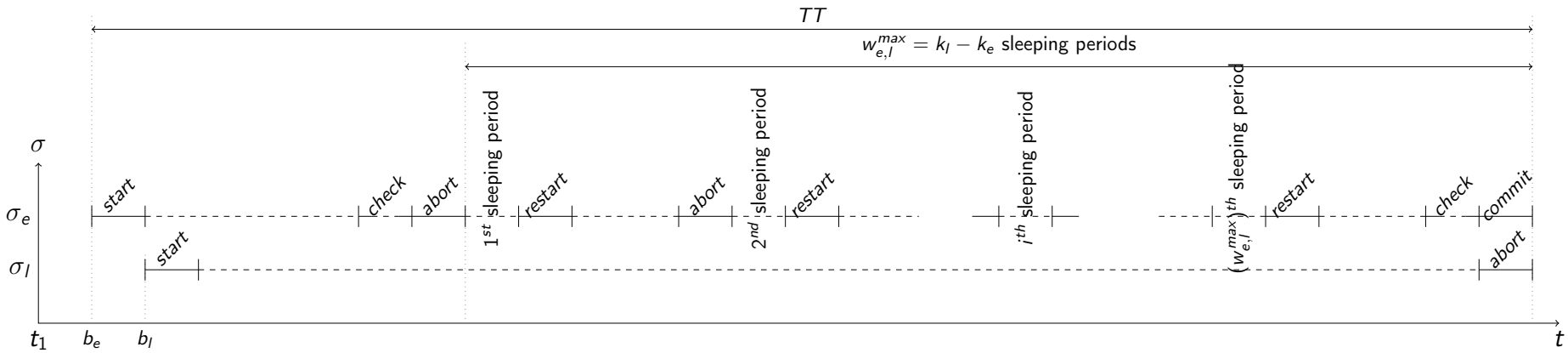
In a system using a software transactional memory whose contention manager is Polka, whose conflict detection is *lazy* and in which all transactions are timely correct, the maximum period needed by a recurring aborted transaction σ_l to reach the maximum possible karma k_S^{max} is given by :

$$\Delta_l^{max} = \Big(\Big\lfloor \frac{k_S^{max}}{2} \Big\rfloor + 1\Big) \times TT$$

Polka : a Wait-free contention manager

Theorem 5.5

In a system using a software transactional memory, which contention manager is Polka, and which conflict detection is lazy, if every transaction is timely correct, then *every transaction progresses*.



Proof (*cases (ii.d.3) and (ii.d.4)*)
$\exists \sigma_e \in Enemies_l : \xi_e = Checking \wedge k_e \leq k_l$

σ_l is necessarily *timely correct* : σ_e will ultimately abort σ_l when reaching the maximum number of sleeping periods.

Plan

- TM allow programmers to embed a sequence of operations applied to shared resources into *a transaction*.
- *Contention managers* allow some transactions *to commit* and force other conflicting ones *to abort* BUT they don't provide any upper bounds on the execution time of transactions
- Focus on Polka contention manager for use in hard real-time systems
- Our contributions :
  - An in-depth timing analysis of Polka contention manager
  - Upper bounds on both the execution time and the number of abortions of transactions induced by the Polka contention manager.
  - A formal proof of the *wait-free* progress guarantees of Polka contention policy, provided all transactions are timely correct.