WTTM 2015

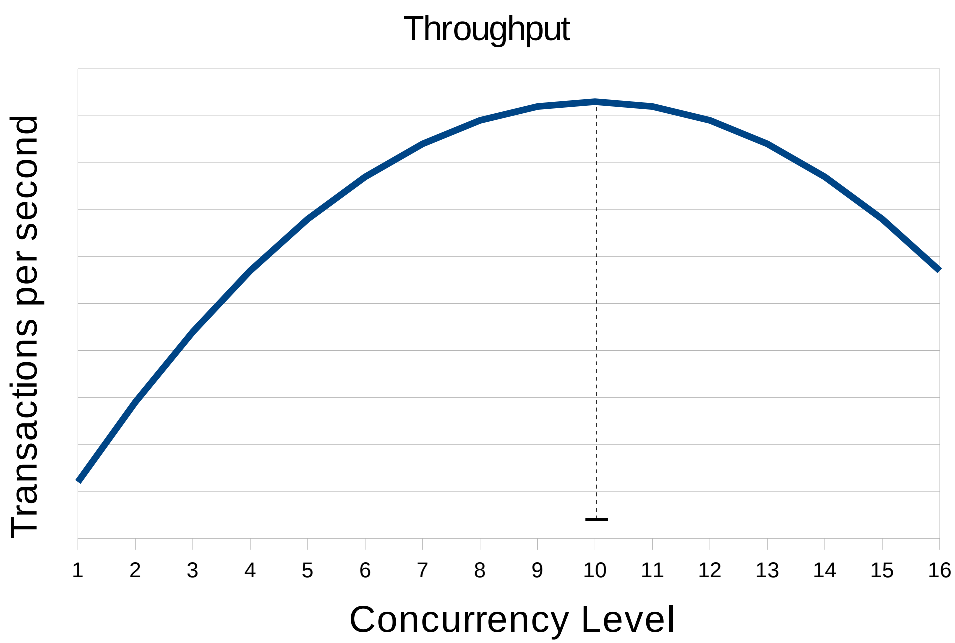
7th Workshop on the Theory of Transactional Memory

On Exploring Markov Chains for Transaction Scheduling Optimization in Transactional Memory

Pierangelo Di Sanzo, Marco Sannicandro,
Bruno Ciciani, Francesco Quaglia

DIAG, Sapienza University of Rome

Effects of Concurrent Execution of Transactions



Transaction concurrency level too low:
performance is penalized due to limitation of parallelism and underutilization of hardware resources

Optimal transaction concurrency level

Transaction concurrency level too high:
loss of performance due to high **data contention** causing **abort and re-run of transactions**.

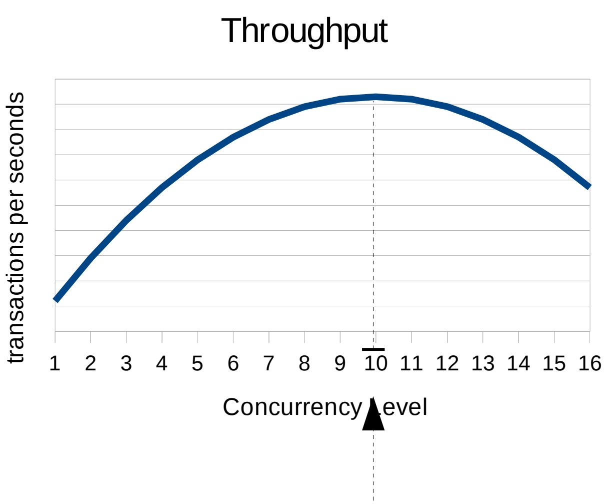
Identifying the optimal concurrency level ...

The **optimal concurrency level depends** on:

- transaction/workload profile (transaction length, data access distribution, read/write ratio, ...)
- hardware architecture

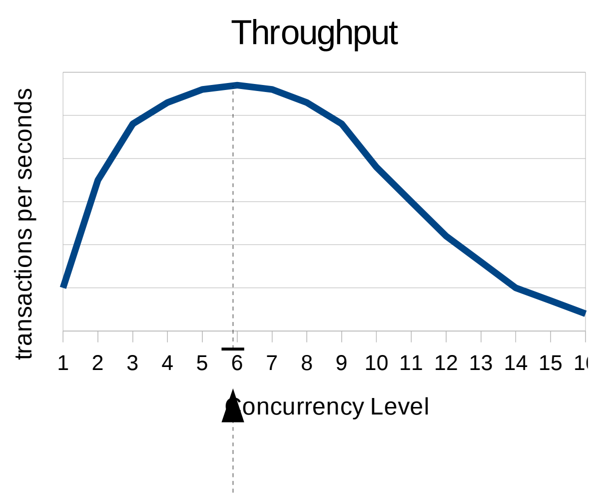
The workload profile may **change** during the execution of the application → the optimal concurrency level may **change** during the execution of the application

Application execution phase 1



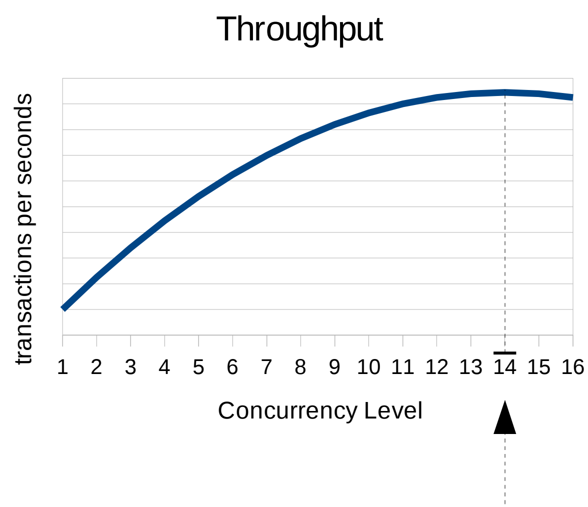
optimal concurrency level: 10

Application execution phase 2



optimal concurrency level: 6

Application execution phase 3



optimal concurrency level: 14

Transaction Scheduling

Transaction Scheduling: Transactions are blocked or allowed to run depending on some scheduling policy

The workload profile may **change** during the execution of the application

<u>Adaptive scheduling approach</u>: the scheduler takes decisions on basis of run-time observations

Transaction Scheduling in TM

Transaction Scheduling policies

Based on system performance prediction models

- Analytical model-based approaches [1]
- Parametric system performance models [2]
- Machine learning-based approaches [3]
- Interpolating functions [4]

Heuristic-based approaches

- Hill-climbing [5]
- Pro-active transaction scheduling: serializing transactions when the abort probability is (estimated to be) high [6,7,8]

Pros and Cons of Performance Prediction-based Approaches

Pros: the ability of predicting the system throughput for different system configurations allows at run-time to quickly “jump” to the best scheduler configuration

Cons: these approaches require ***a priori analysis phases*** *(e.g. training phases)*, during which various parameters have to be measured while running the application with different workload profiles and/or different scheduler configurations (in terms of, e.g., number of admitted transactions).

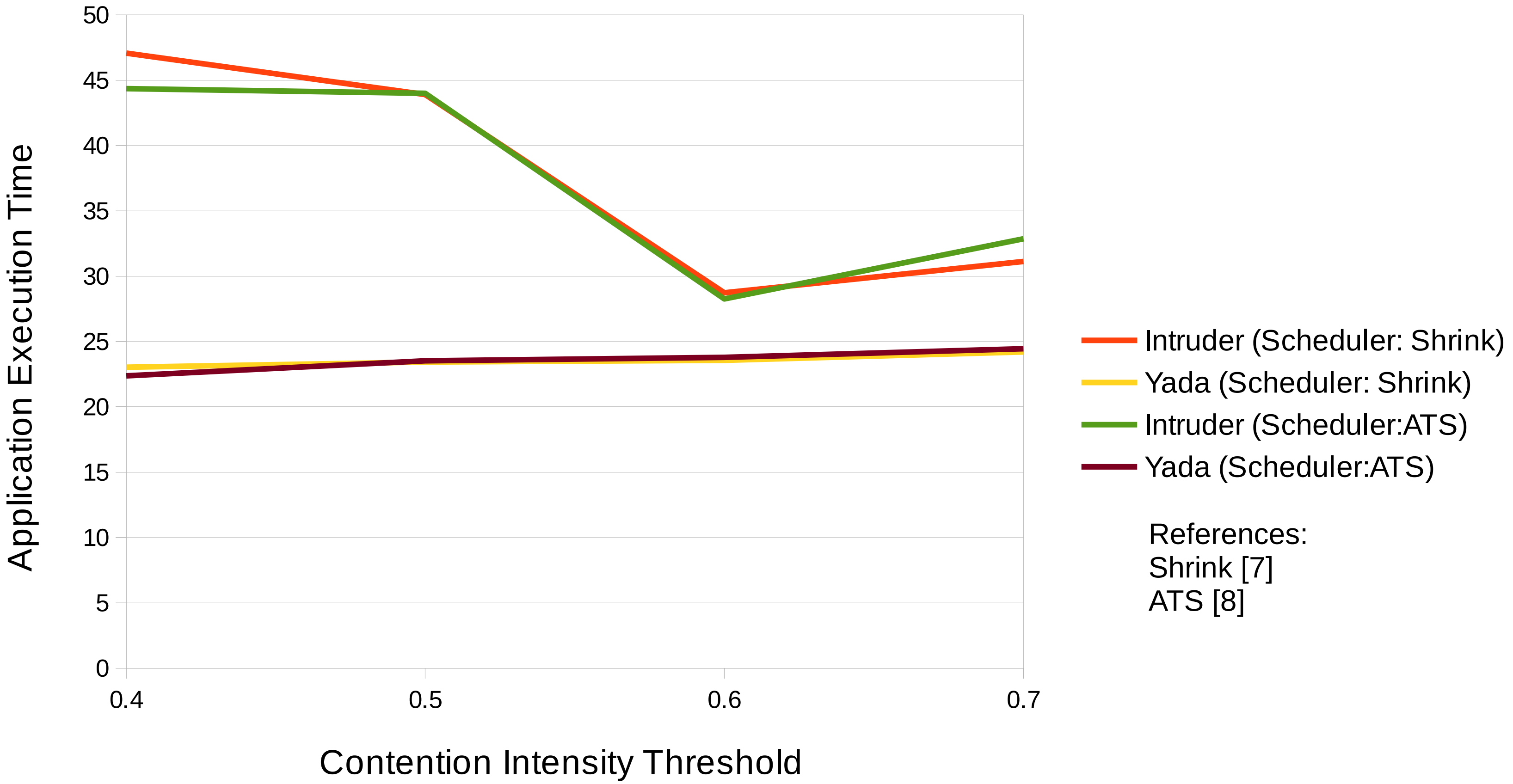
Pros and Cons of Heuristic-based Approaches

Pros: no a priori analysis phases of the application are required

Cons:

- the optimal solution is not guaranteed
- time to converge
- the user has to configure some parameters (e.g. conflict rate/abort probability thresholds) on the basis of which the scheduler takes decisions

Effects of Different Thresholds of Scheduling Algorithms



References:
Shrink [7]
ATS [8]

Hardware configuration:
16-cores HP ProLiant server, equipped with 2GHz AMD Opteron 6128 processors, 64 GB of RAM and the Linux operating system (kernel version 2.7.32-5-amd64).

Application configuration:
Intruder - input: -a10 -l128 -n262144
Yada - input: -a15 -i yada/inputs/ttimeu1000000.2

Getting the best of the two worlds ...

A system performance model ...

· for predicting the system throughput depending on the concurrency level, ...

· that can be instantiated on-the-fly (no a priori observation phases required), …

· easy to be re-configured when the workload profile changes, ...

and

· that does not require a “skilled” user for setting up the optimal scheduler configuration.

Target System Model

- ***N*** running threads

- each thread can execute both **transactions** or **non-transactional code (*ntc*) blocks.**

- a transaction is aborted and restarted upon conflict

- **Transaction scheduling policy:**

  - the scheduler accepts at most ***m*** (with $m \leq N$) concurrent transactions (other transactions are blocked)

  - a blocked transaction is allowed to run when another running transaction commits.

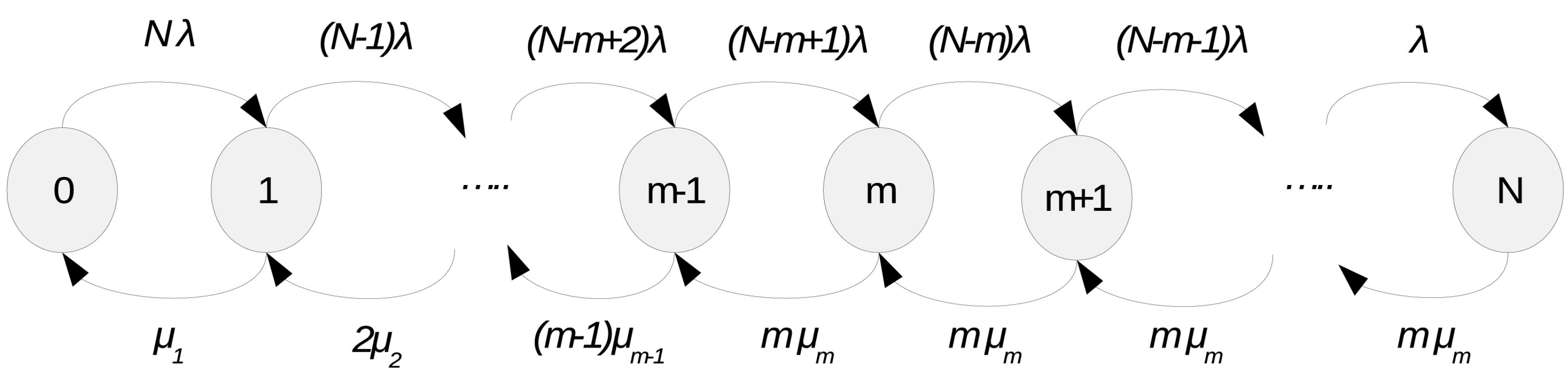
A lightweight Markov Chain-based Performance Prediction Model

Modelling the system behaviour through a set of *states* and *states transitions*

Continuous-time homogeneous Markov Chain (CTMC) with ***N*** states (finite state space)

· state ***k*** of the CTMC represents a state of the system when there are ***k*** threads executing transactions (both running or blocked transactions)

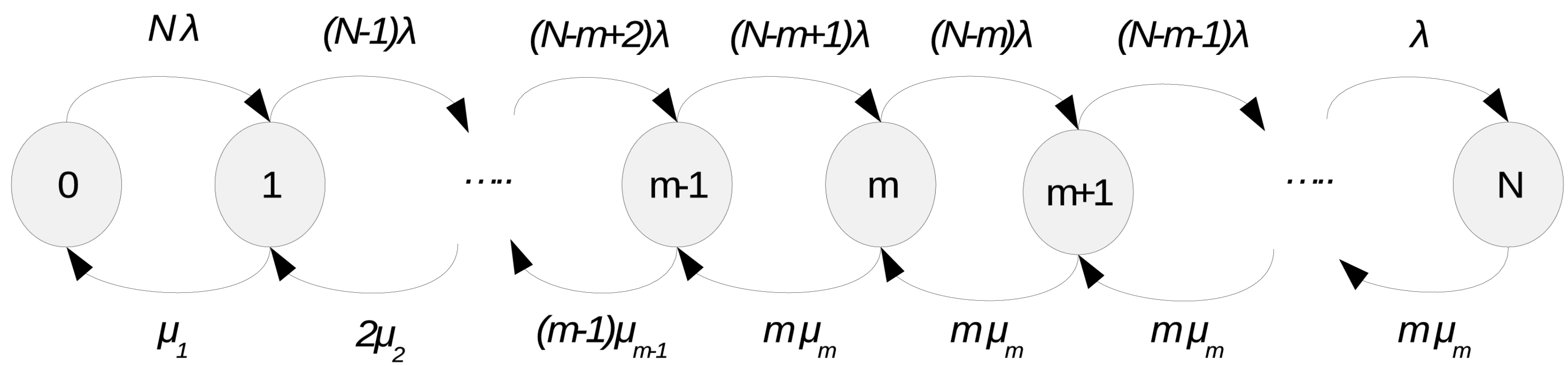
→when the system is in state ***k*** there are ***N-k*** threads executing *ntc* blocks.



A lightweight Markov Chain-based Performance Prediction Model

· A *transition* from state ***k*** to ***k+1*** occurs when a threads starts a new transaction

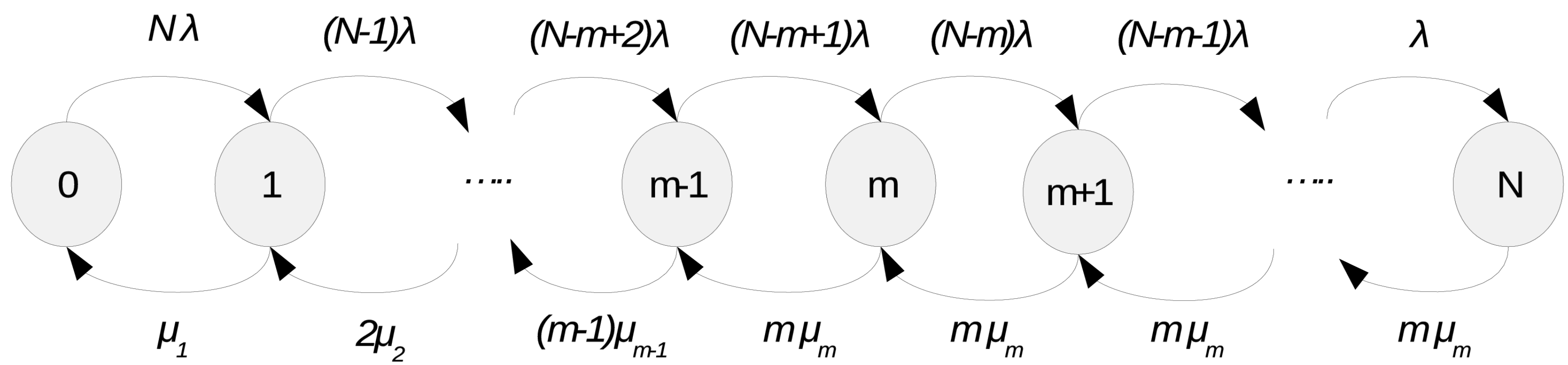
· A *transition* from state ***k*** to state ***k-1*** occurs upon the commit of whichever running transaction



A lightweight Markov Chain-based Performance Prediction Model

t$_{\textbf{ntc}}$: *average execution time* of *ntc blocks* →
inter-arrival rate of transactions along any thread $\lambda = \frac{1}{t_{ntc}}$

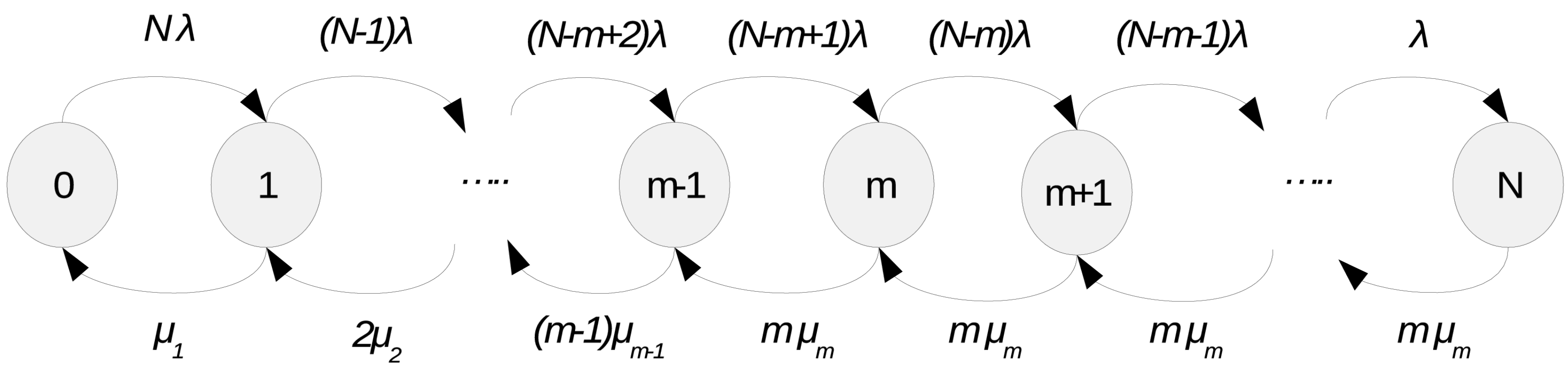
→ *transition rate* from state ***k*** to ***k+1***: $\lambda_k = (N - k) \cdot \lambda$



A lightweight Markov Chain-based Performance Prediction Model

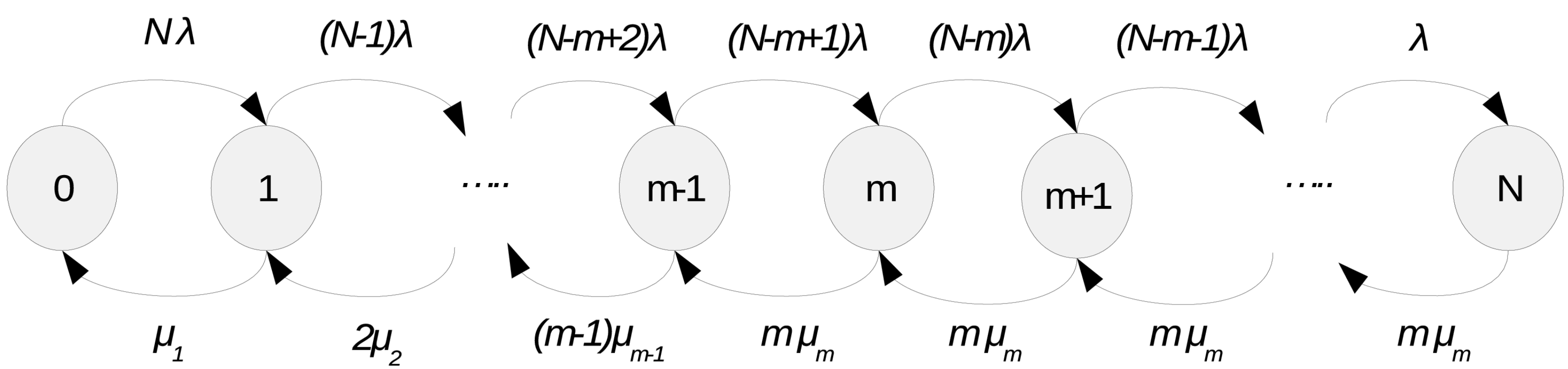
t_k: *average transaction execution time* when there are ***k*** executing transactions →

transaction execution rate of a thread for state ***k***: $\mu_k = \frac{1}{t_k}$



A lightweight Markov Chain-based Performance Prediction Model

· for any state ***k*** **≤** ***m***, since exactly ***k*** transactions are running (i.e. none is blocked), the t*ransition rate* from state ***k*** to state ***k* − *1*** is $\gamma_k = k \cdot \mu_k$



· for any state ***k*** **>** ***m***, the there are ***m*** running transactions (the other ***k* − *m*** transactions are blocked →

for all states such that ***k*** **>** ***m***, the *transition rate* from the state ***k*** to the state ***k* − *1*** is $\gamma_k = m \cdot \mu_k$

A lightweight Markov Chain-based Performance Prediction Model

Transaction execution time when the system is in state ***k***:

$$t_k = w_{t,k} + u_{t,k}.$$

Average time to execute all aborted runs (*wasted time*) of a transaction

Average time to execute the last run of a transaction (*useful time*), i.e. the run that successfully commits

$$w_{t,k} = w^r_{t,k} \cdot r_k$$

average time to execute an aborted transaction run

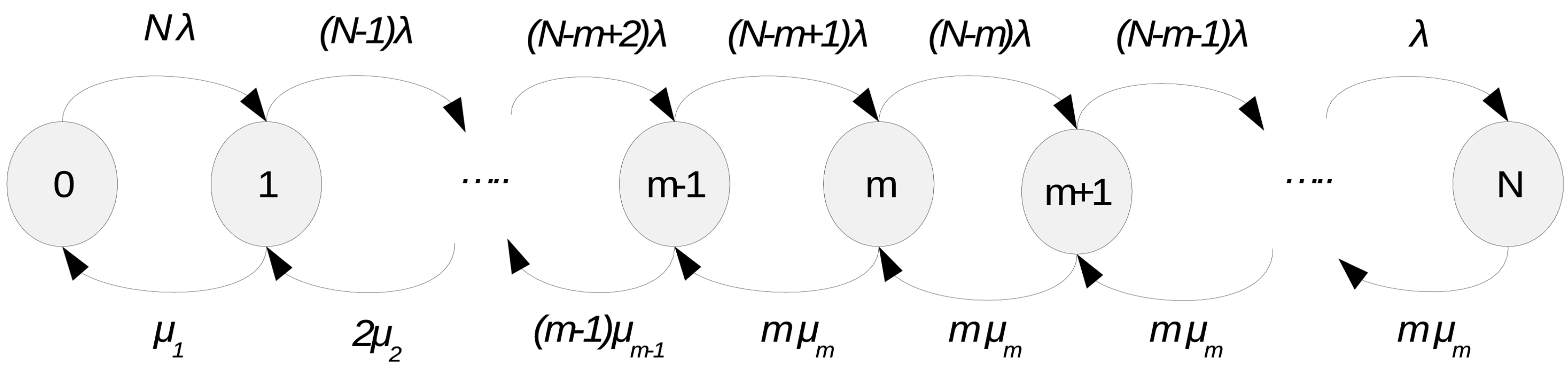
average number of aborts (or re-runs) of a transaction

$$r_k = \frac{p_k}{1 - p_k}$$

p_k : transaction abort probability for the state ***k***

A lightweight Markov Chain-based Performance Prediction Model

{q_k:0<=k<N} *stationary probability vector* of the CTMC



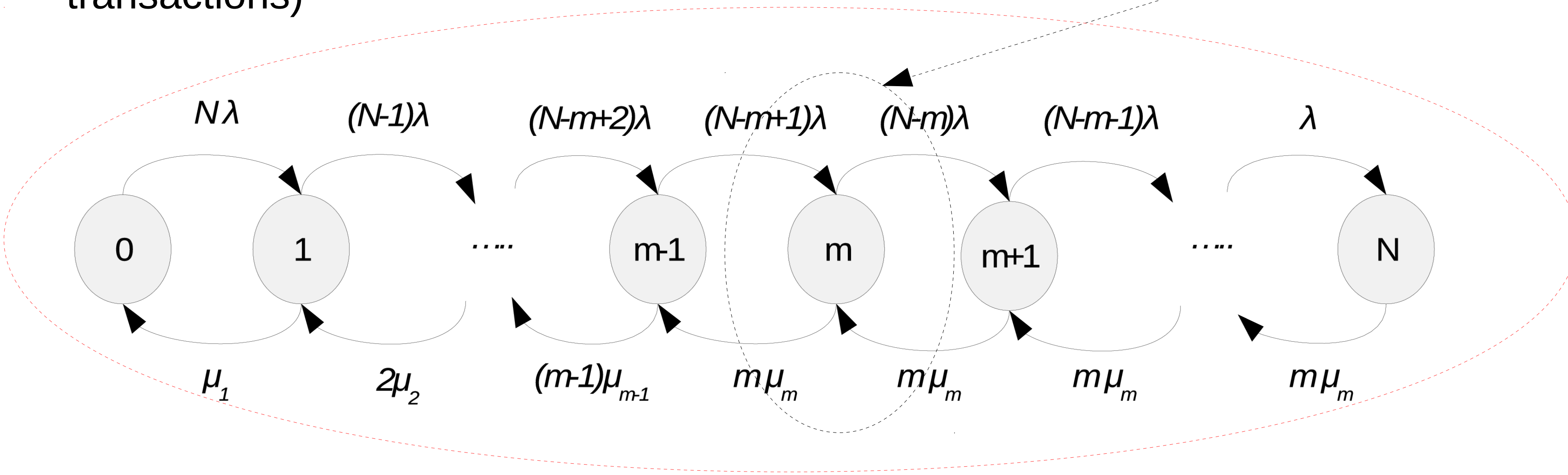
System throughput $\boldsymbol{thr_m}$ when the scheduler admits at most ***m*** running transactions:

$$thr_m = \sum_{i=1}^{N} q_k \cdot \gamma_k$$

Model Instantiation

To instantiate the model, values of parameters $u_{t,k}$, $w^r_{t,k}$, t_{ntc} and p_k have to be known.

They can be can be measured, for each ***k,*** by observing the system running with a **fixed scheduler configuration** (i.e., admitting at most ***m*** concurrent transactions)



What-if Analysis

Once above parameters have been measured, we can use the model for predicting the system throughput for **different scheduler configurations** (i.e. for different values of ***m****)*. This can be done by modifying the value of ***m*** of the CTMC and solving the model.

Experimental Evaluation: model validation with STAMP and TinySTM

Model validation

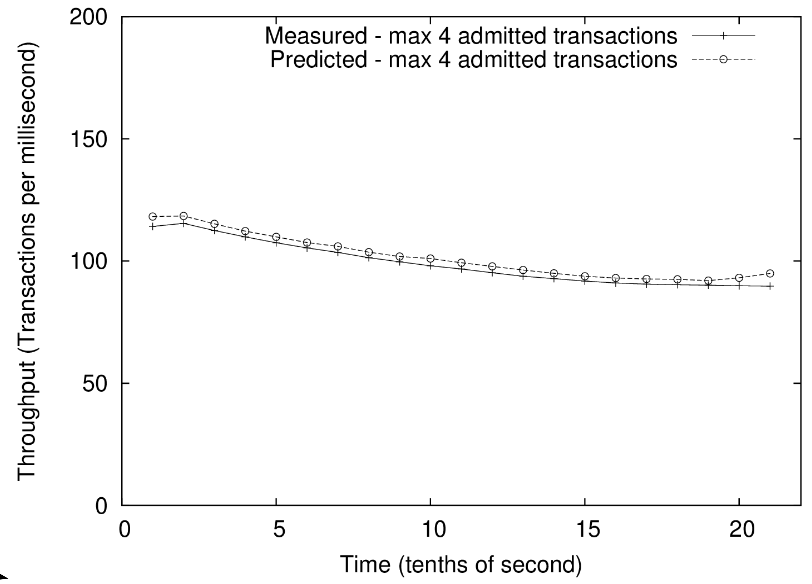
- 8 threads
- scheduler configuration: **m=4**
- observation period: 1000 executed transactions

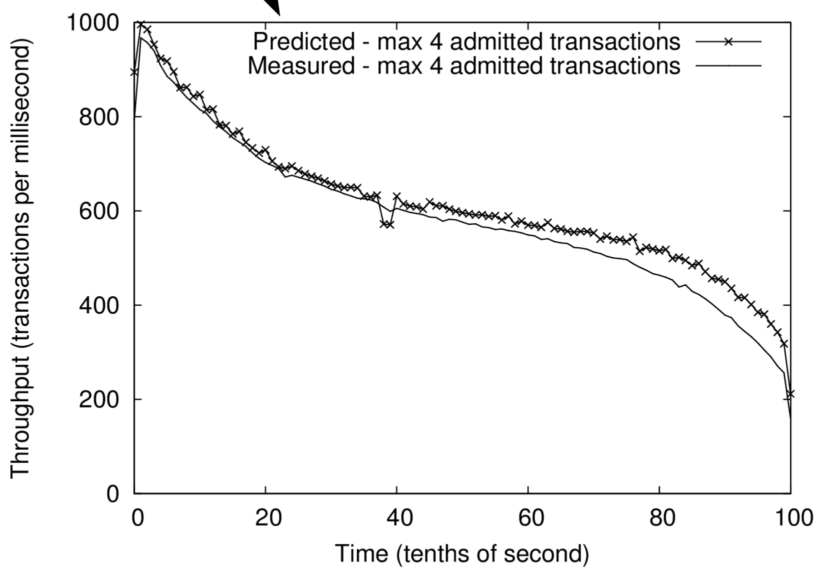
Average Relative Prediction Error:

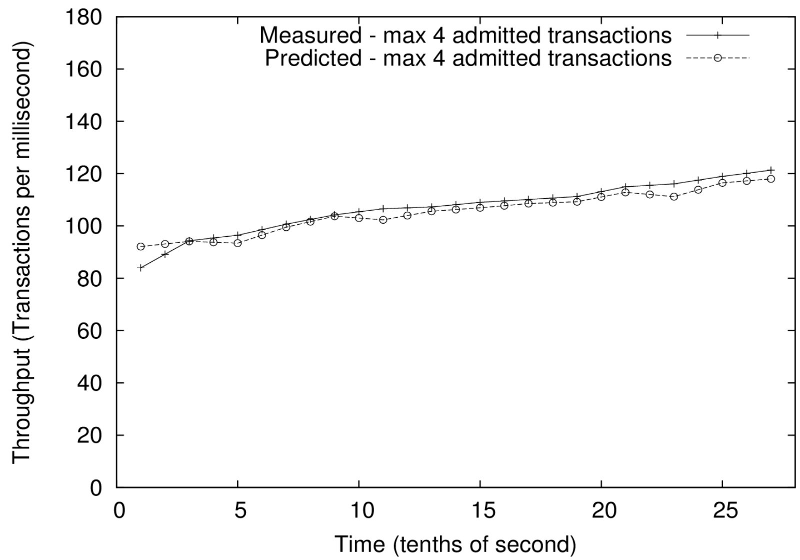
Vacation 2.7%

Yada 2.5%

Intruder: 8.9%







What-if Analysis with STAMP and TinySTM

What-if Analysis

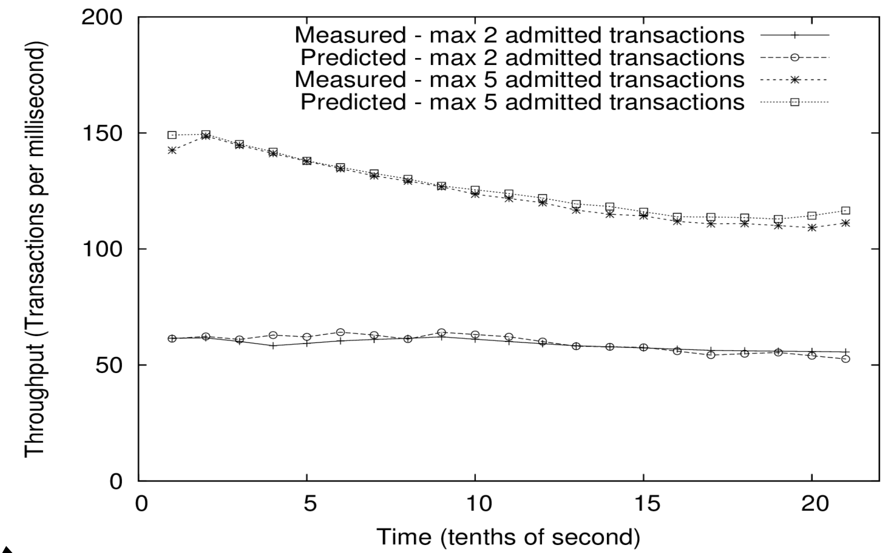
Throughput prediction for m=2 and m=5 based on system observation for m=4

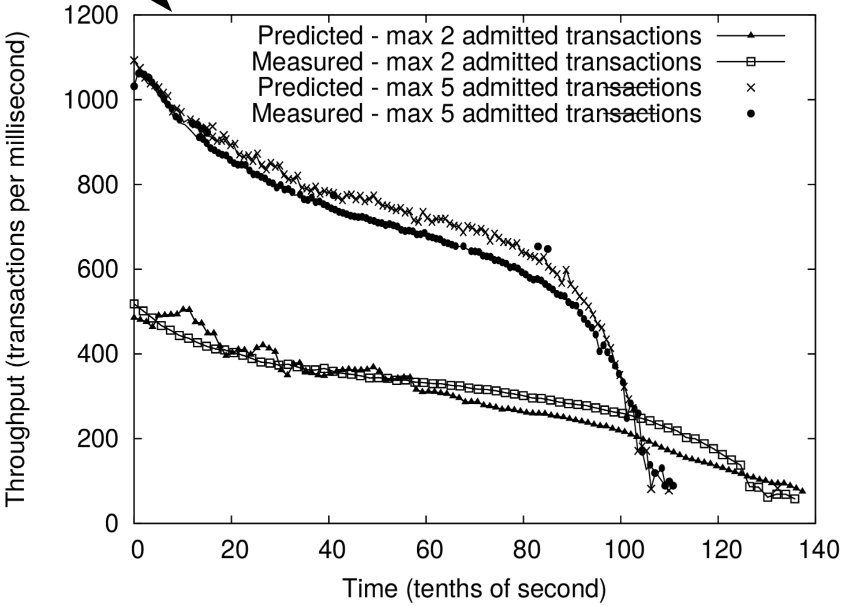
<u>Average Relative Prediction Error:</u>

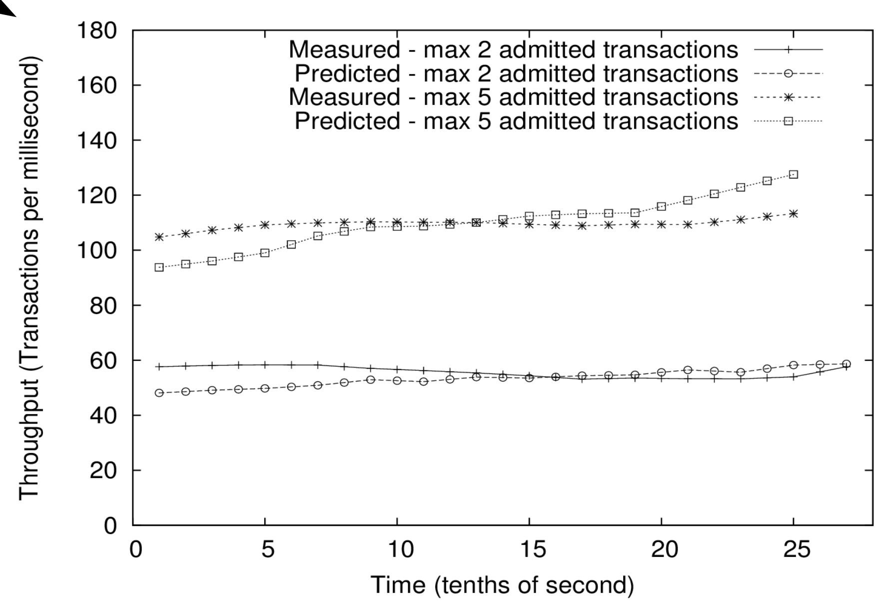
Vacation: 2.6% (m=2), 1.9% (m=5)

Yada: 7.2% (m=2), 5.9 % (m=5)

Intruder: 8.8% (m=2), 9.2% (m=5)







Thank you

References

[1] Diego Rughetti, Pierangelo di Sanzo, Bruno Ciciani, and Francesco Quaglia. Machine learning-based self-adjusting concurrency in software transactional memory systems. In Proc. 20th Int. Symposium on 4Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pages 278–285. IEEE, 2012.
[2] Aleksandar Dragojevi´c and Rachid Guerraoui. Predicting the scalability of an stm a pragmatic approach. In Proc. 5th ACM Workshop on Transactional Computing. ACM, 2010.
[3] Pierangelo Di Sanzo, Francesco Del Re, Diego Rughetti, Bruno Ciciani, and Francesco Quaglia. Regulating concurrency in software transactional memory: An effective model-based approach. In Proceedings of the Seventh IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO), 2013.
[4] Pierangelo Di Sanzo, Francesco Del Re, Diego Rughetti, Bruno Ciciani, Francesco Quaglia Regulating Concurrency in Software Transactional Memory: An Effective Model-based Approach. In Proc. Seventh IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2013)
[5] Diego Didona, Pascal Felber, Diego Harmanci, Paolo Romano, and Joerg Schenker. Identifying the optimal level of parallelism in transactional memory systems. In Proc. International Conference on Networked Systems, NETYS. Springer, 2013.
[6] Mohammad Ansari, Christos Kotselidis, Kim Jarvis, Mikel Luj´an, Chris Kirkham, and Ian Watson. Advanced concurrency control for transactional memory using transaction commit rate. In Proc. 14Th Int. Euro-Par Conference on Parallel Processing, pages 719–728. Springer-Verlag, 2008.
[7] Aleksandar Dragojevi´c, Rachid Guerraoui, Anmol V. Singh, and Vasu Singh. Preventing versus curing: avoiding conflicts in transactional memories. In Proc. 28th ACM Symposium on Principles of Distributed Computing, pages 7–16. ACM, 2009.
[8] Richard M. Yoo and Hsien-Hsin S. Lee. Adaptive transaction scheduling for transactional memory systems. In Proc. 20th Symposium on Parallelism in Algorithms and Architectures, pages 169–178. ACM, 2008.