Seventh IEEE International Conference on
Self-Adaptive and Self-Organizing Systems

SASO 2013

Regulating Concurrency in Software Transactional Memory: An Effective Model-based Approach

Pierangelo Di Sanzo, Francesco Del Re, Diego Rughetti,
Bruno Ciciani, Francesco Quaglia

DIAG, Sapienza University of Rome

SAPIENZA
UNIVERSITÀ DI ROMA

HPDCS
Research Group

Recent Trend in Computer System Architectures

Nowadays **multi-core/processor systems** have become mainstream platforms

Hardware performance continues to improve mainly due to:

- even more cores in a single processor
- even more processors in a single machine

…

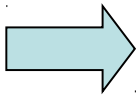
Single-threaded/process applications can not take advantage of such a hardware performance improvement.

need for
multi-threaded/process applications

The Synchronization Problem

The **synchronization problem** in concurrent applications: code sections accessing shared data may have to be synchronized (e.g. critical sections)

- using traditional synchronization techniques (i.e. locks, semaphores, monitors, …) is not easy
- fine-grained synchronization is a time-consuming task

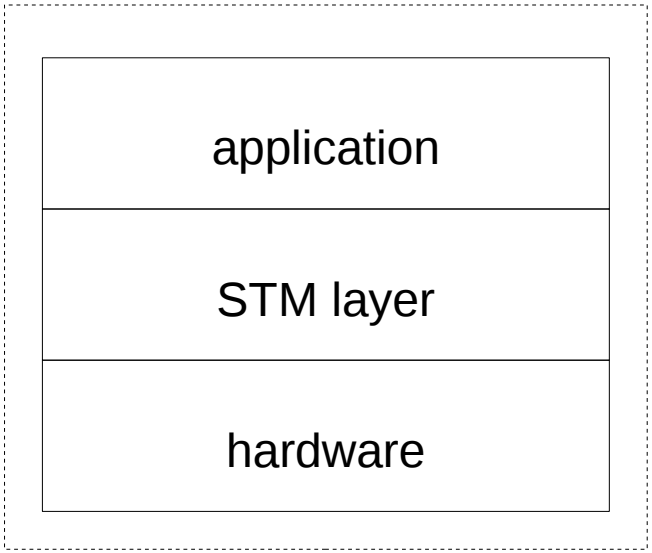


- pitfalls for programmers: deadlocks, livelocks, priority inversions, code composition is complex, scalability issues, …
- debug is complex

Transactional Memories (TM):

- programming paradigm for multi-core/processor systems
- simplifies the development of parallel and concurrent applications

key idea: hide away synchronization issues by using transactions



Transactional Memories

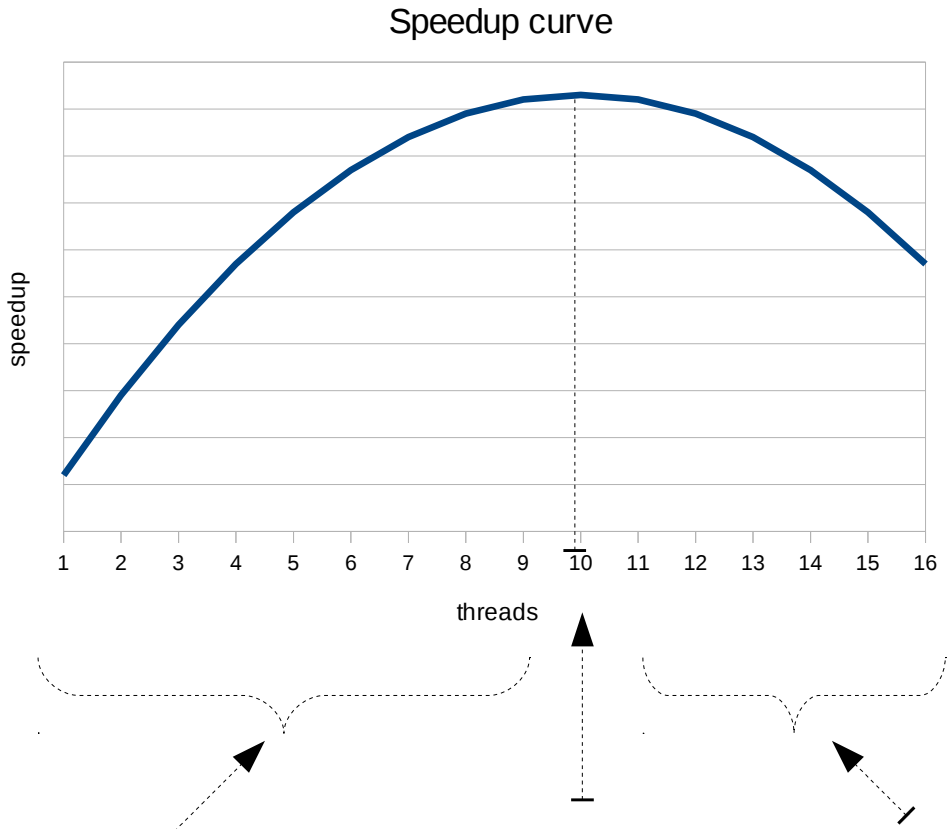
Using TM: code example (queue pop)

transaction

```
…
elem_t* elemPtr;
…
TM_BEGIN();

long pop      = (long)TM_READ(queuePtr->pop);
long push     = (long)TM_READ(queuePtr->push);
long capacity = (long)TM_READ(queuePtr->capacity);

long newPop = (pop + 1) % capacity;
if (newPop == push) {
    elemPtr = NULL;
    } else {
        void** elements = (void**)TM_READ(queuePtr-> elements);
        elemPtr = (pair_t*) TM_READ(elements[newPop]);
        TM_WRITE(queuePtr->pop, newPop);
}

TM_END();
…
…
```

The underlying transactional memory layer takes care of ensuring atomic and isolated executions of transactions

Transactional Memories: How Many Threads?



<u>concurrency level too low:</u>
performance is penalized due to limitation of parallelism and underutilization of hardware resources

<u>optimal concurrency level</u>

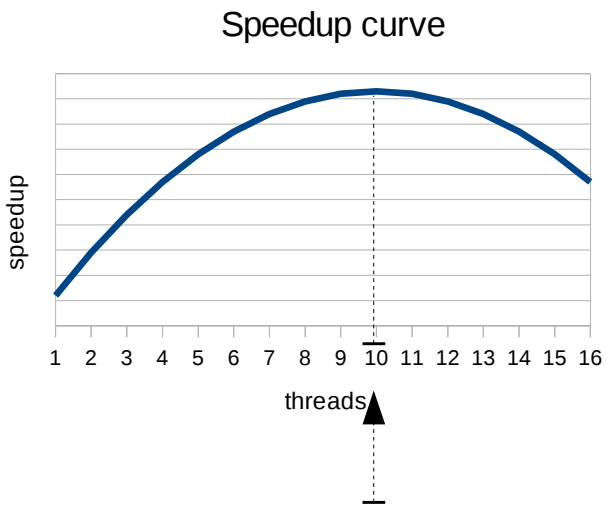
<u>concurrency level too high:</u>
loss of performance due to excessive **data contention** and consequent **transaction aborts and re-runs**.

Identifying the optimal concurrency level...

The optimal concurrency depends on:
application logic, workload profile, hardware architecture, ...

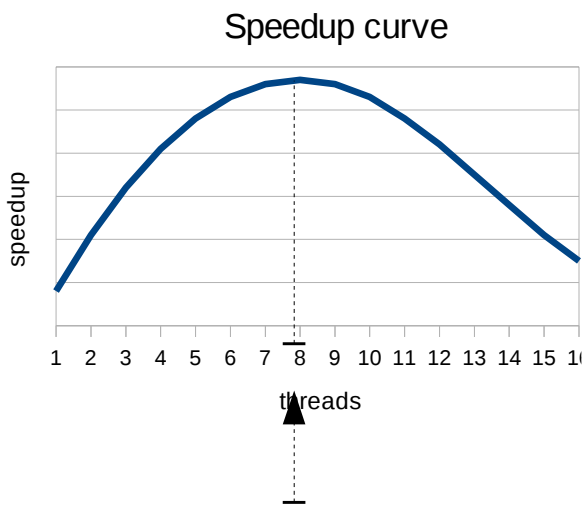
Additionally, the optimal concurrency level may change depending on the application execution phase.

phase 1



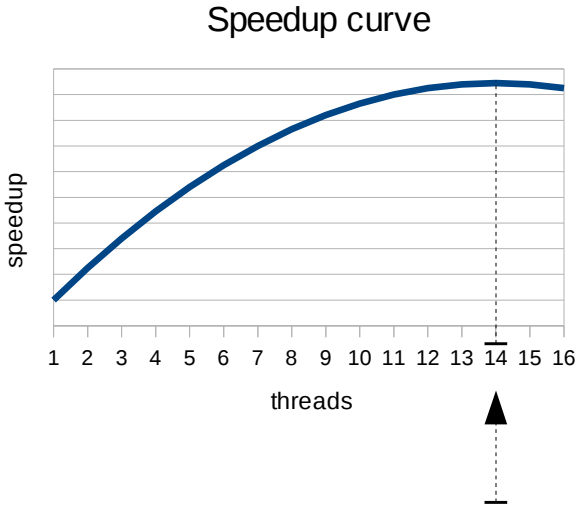
optimal concurrency level: 10

phase 2



optimal concurrency level: 8

phase 3



optimal concurrency level: 14

Goal and Proposed Solution

Goal:

enabling TM platforms to self-regulate the concurrency level

How:

1) a parametric performance model of TM applications is used to estimate the throughput of an application as a function of the:

- concurrency level (number of concurrent threads)
- the current workload profile of the application

2) regression analysis is exploited to customize the parametric performance model for a specific TM system (application + hardware platform)

3) a controller is integrated with the TM platform. At run-time, the controller exploits the customized model in order to decide, on basis of the observed workload profile, the number of concurrent threads to keep active

The TM Parametric Performance Model

The parametric performance model estimates the transaction abort probability of transaction p_a as a function of:

- the average size of the transaction read-set (rs_s)
- the average size of the transaction write-set (ws_s)
- the average execution time of committed transaction runs (t_t)
- the average execution time of code blocks outside of transactions (t_{ntc})
- the read/write affinity (rw_a, i.e. the probability that an object read by a transaction is also written by other transactions)
- the write/write affinity (ww_a, i.e. the probability that an object written by a transaction is also written by other transactions)
- the number of concurrent threads (k)

$$p_a = f(rs_s, ws_s, rw_a, ww_a, t_t, t_{ntc}, k)$$

Model Construction Approach

Results from analytical modeling studies of transactional systems (e.g. [1,2]) :

$$p_a = 1 - e^{-\alpha}$$

where α is a complex function which is hard to identify unless making strong assumptions on workload profile, operations' execution speed, data contention, etc

may cause high prediction error with real applications

Model Construction Approach

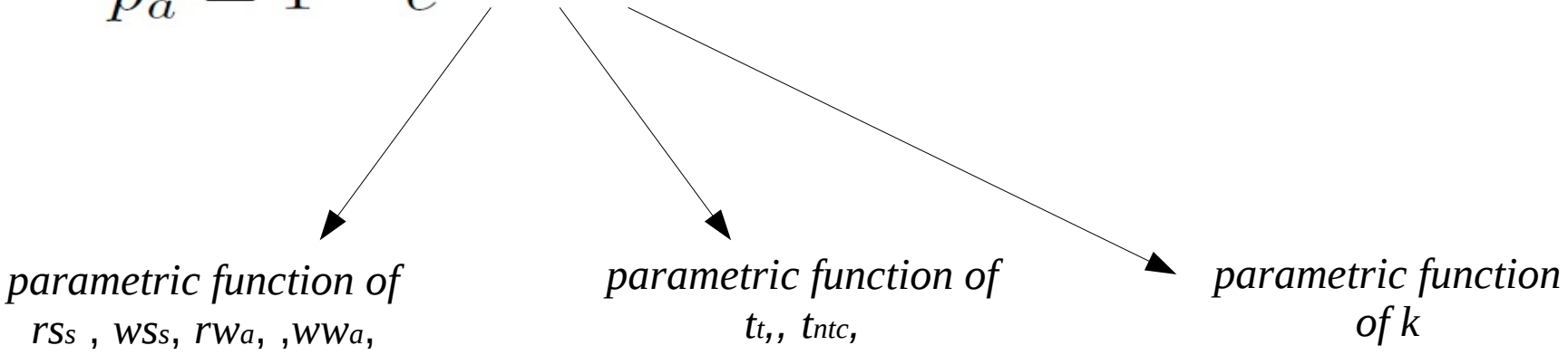
Proposed approach:

- Simulation has been used to determine a parametric expression which captures the shape of the curve of the function p_a depending on the the workload profile

- The parametric expression has been validated using data achieved by executing TM applications on real systems

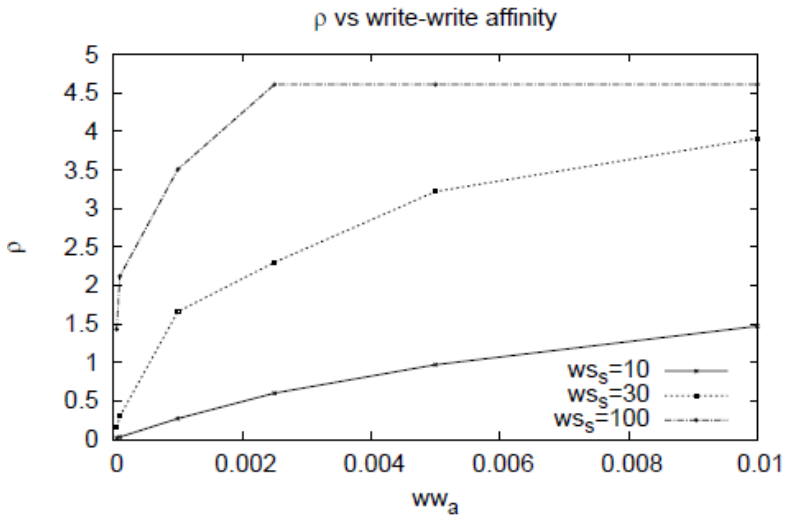
Basic equation:

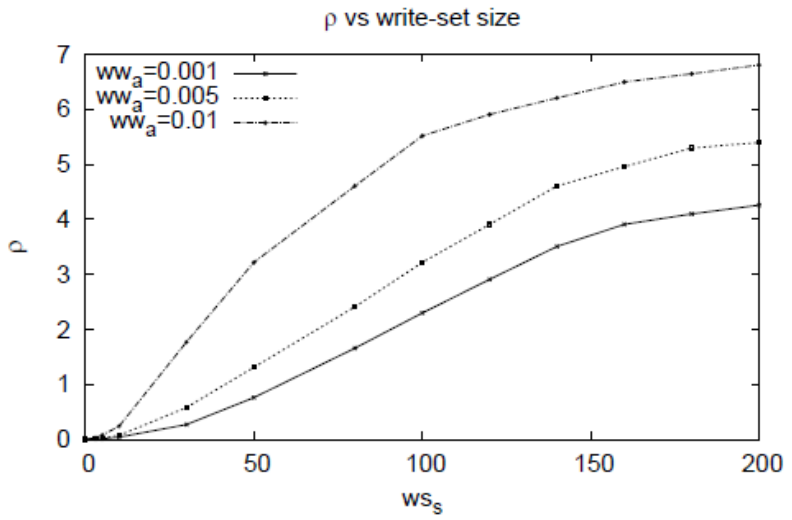
$$p_a = 1 - e^{-\rho \cdot \omega \cdot \phi}$$



Model Construction Approach

Case of ρ: expressing ρ as a function of ws_s and ww_a





fitting function:

$$[c \cdot (ln(b \cdot ws_s + 1)) \cdot ln(a \cdot ww_a + 1)]^d$$

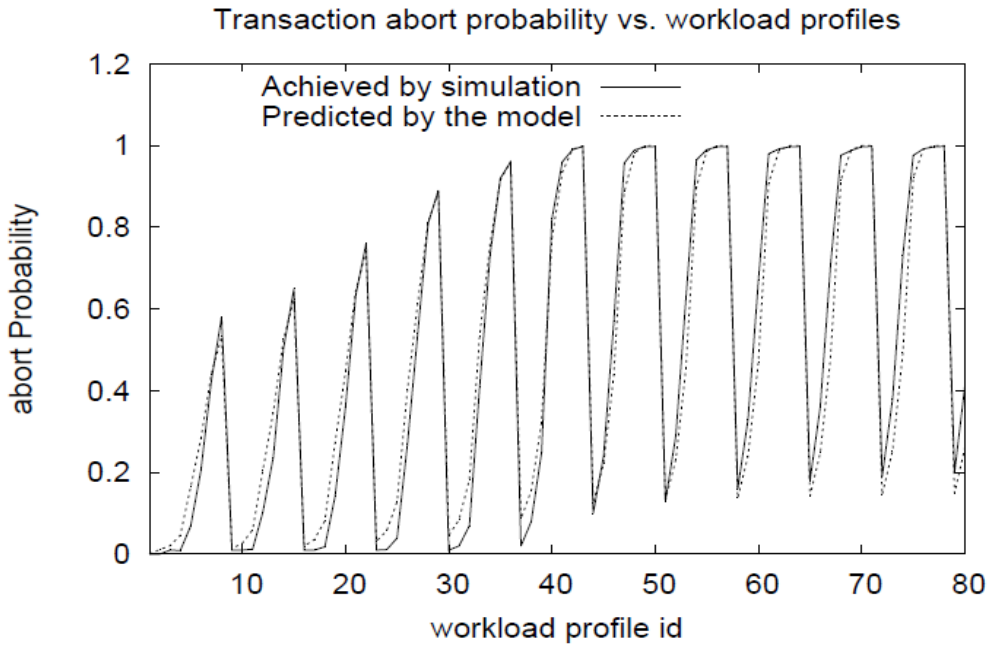
fitting parameters: a, b, c, d

Model Construction Approach

fitting function: $[c \cdot (ln(b \cdot ws_s + 1)) \cdot ln(a \cdot ww_a + 1)]^d$

Error evaluation with respect simulation data:

- fitting parameters (c, b, a, d) calculated through regression analysis (using 40 randomly selected workload profiles)
- average error (using 80 randomly selected workload profiles while varying ww_a and ws_s): 5.3%



Model Construction Approach

The same approach for *ω* and *Φ*, ...

Finally

$p_a = 1 - e^{-\rho \cdot \omega \cdot \phi}$, *where:*

expression of ρ:

$$[c \cdot (ln(b \cdot ws_s + 1)) \cdot ln(a \cdot ww_a + 1)]^d + \\ + [e \cdot (ln(f \cdot rw_a + 1)) \cdot ln(g \cdot rs_s + 1) \cdot ws_s]^z$$

expression of ω:

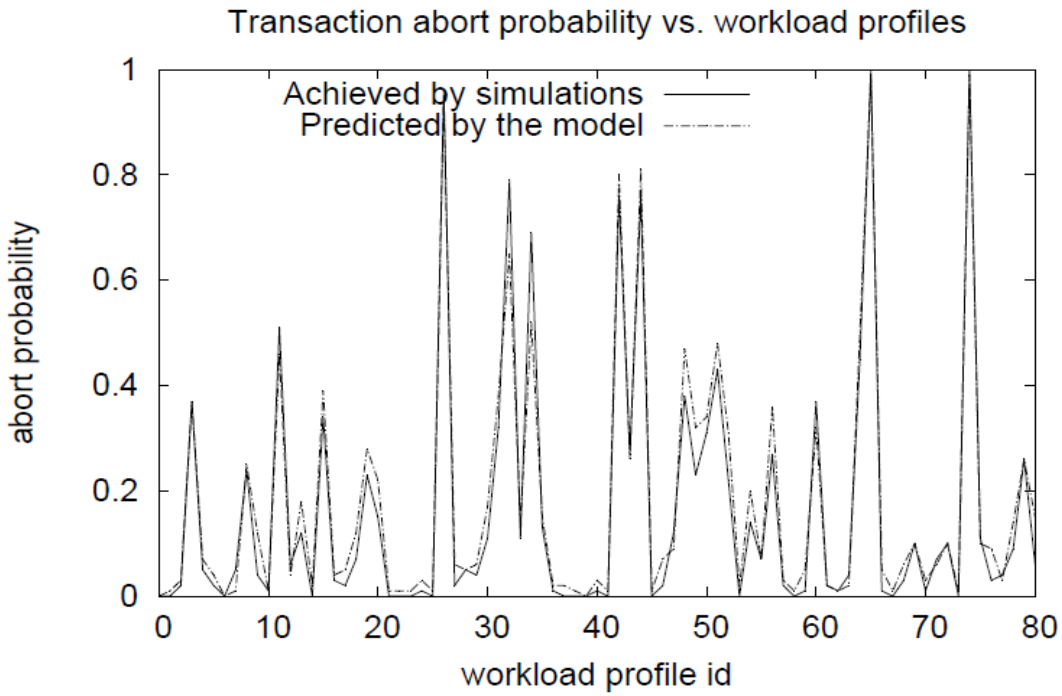
$$h \cdot (ln(l \cdot (k - 1) + 1)$$

expression of Φ:

$$m \cdot ln(n \cdot \theta + 1), \text{ where } \quad \theta = \frac{t_t}{t_t + t_{ntc}}$$

Model Construction Approach

Final average error with respect to simulation data: 4.8%



Model Validation with a Real System

- Evaluation of the prediction error using STAMP benchmark suite [3] and TinySTM [4]

Hardware: HP ProLiant server with 2 AMD OpteronTM6128 Series Processor, 8 cores per CPU (for a total of 16 cores), 32 GB RAM, Linux kernel version 2.7.32-5-amd64.

- Evaluation of the extrapolation capability of the model: for each application, three regression analysis have been performed using three different sets of measurements. Each set of measurements includes 80 samples gathered observing the application running with:

  - 2 and 4 concurrent threads (first set)
  - 2,4 and 8 concurrent threads (second set)
  - 2,4, 8 and 16 concurrent threads (third set)

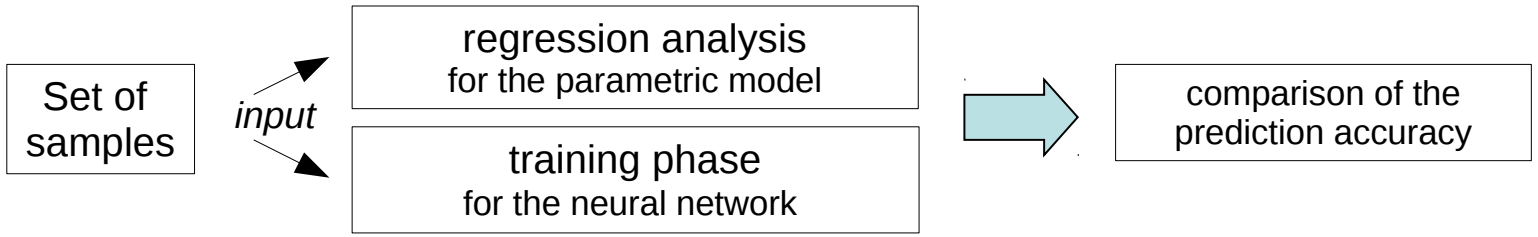
Results (while varying application workload profiles and the number of threads between 2 and 16)

abort probability prediction error (variance in brackets)

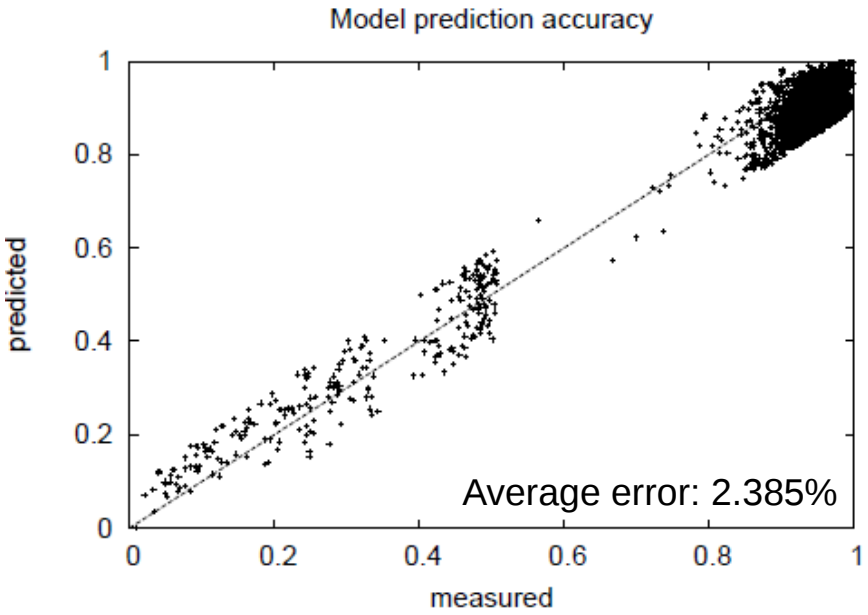
	Observed concurrency levels for the regression analysis		
application	**2/4 threads**	**2/4/8 threads**	**2/4/8/16 threads**
Vacation	2.166% (0,00089)	1.323% (0,00028)	1.505% (0,00032)
Kmeans	18.938% (0,09961)	2.086% (0,00100)	2.591% (0,00109)
Yada	2.385% (0,00029)	2.086% (0,00016)	2.083% (0,00022)

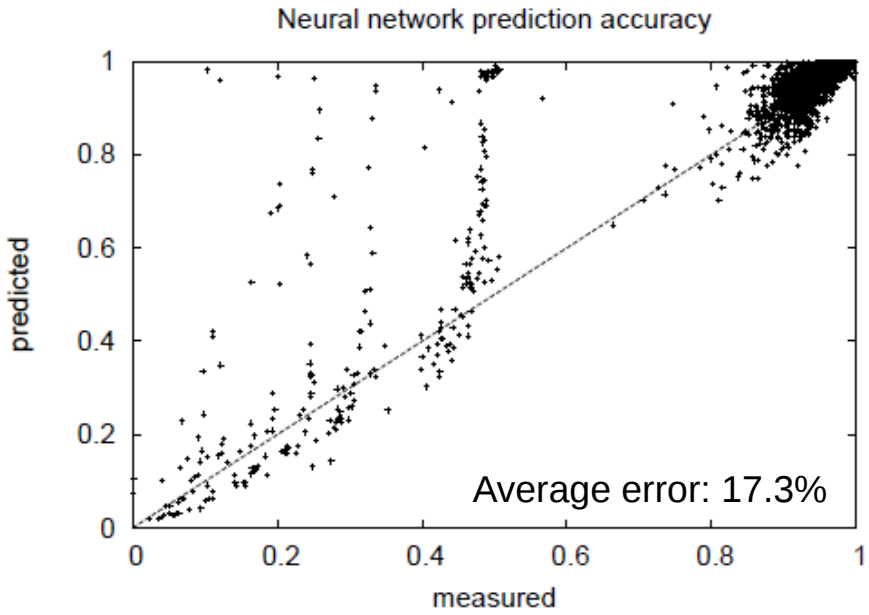
Comparison with a Neural Network-based Performance Model

Evaluation of the extrapolation capability with respect a neural network-based model (proposed in [5])



Results for Yada benchmark (using the first set of samples)

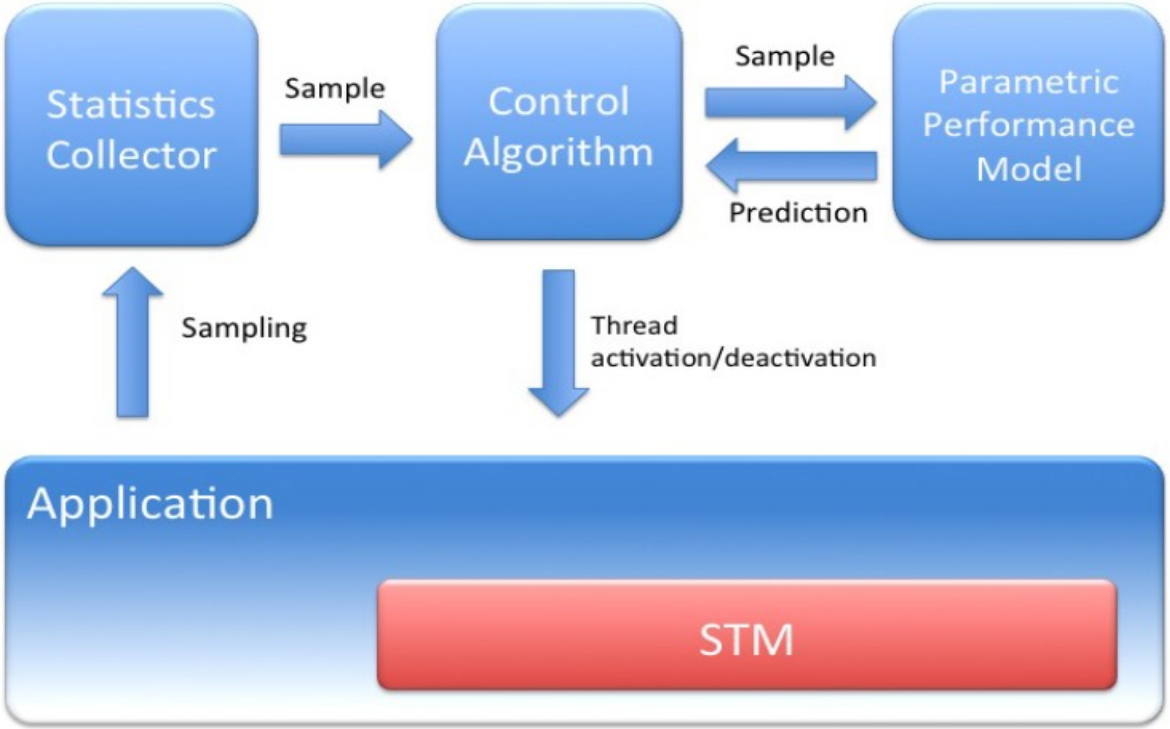




Enabling STM to self-regulate the concurrency level:

architecture of CSR-STM (Concurrency Self-Regulating STM)

available at URL http://www.dis.uniroma1.it/hpdcs/CSR-STM.tar



Periodically, the controller uses the performance model to calculate the expected abort probabilities $p_{a,k}$ while varying the number of concurrent threads k, i.e.:

$$\{(p_{a,k}), 1 \leq k \leq max_{thread}\}$$

hence, the controller keeps active m threads, where m is the value of k such that the application throughput, i.e.

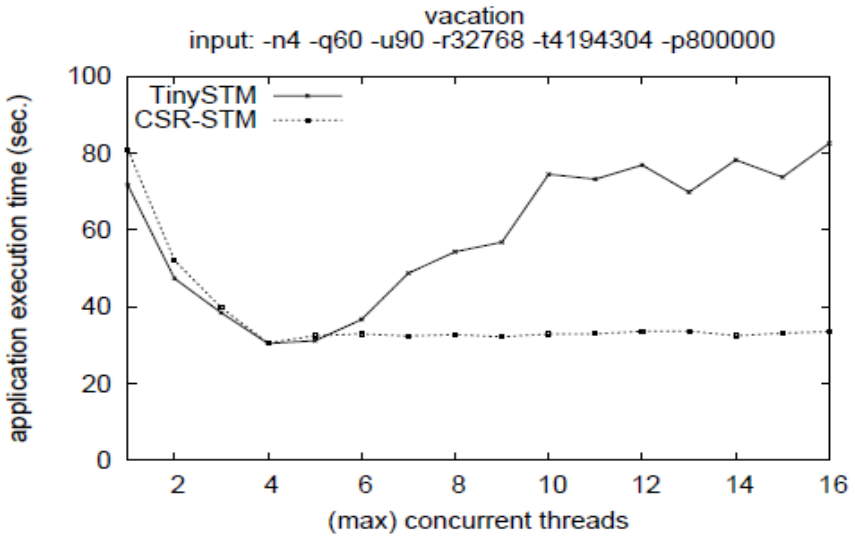
$$\frac{k}{w_{t,k} + t_{t,k} + t_{ntc,k}}$$, is maximized

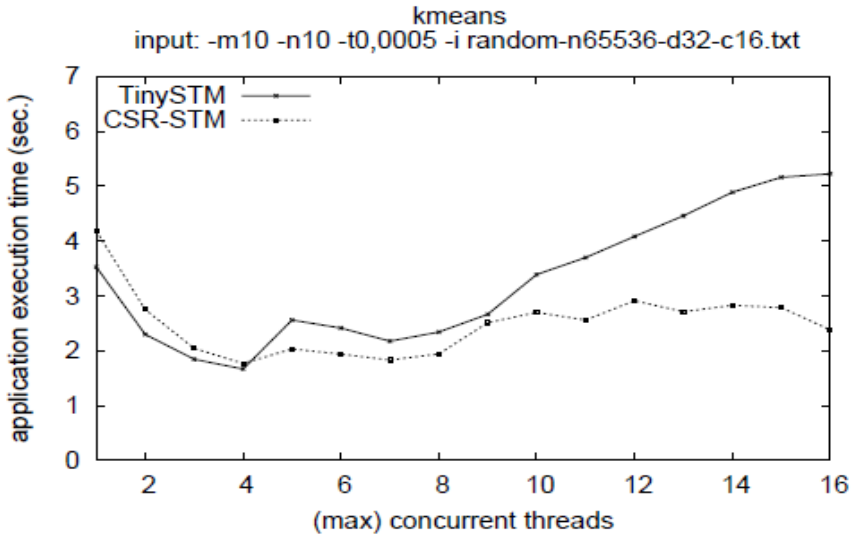
- $w_{t,k}$: average transaction wasted time:
  - It can be calculated using the average number of transaction re-runs: $p_{a,k}/(1-p_{a,k})$
- $t_{t,k}$: average execution time of committed transaction runs
- $t_{ntc,k}$: average execution time of code blocks outside of transactions

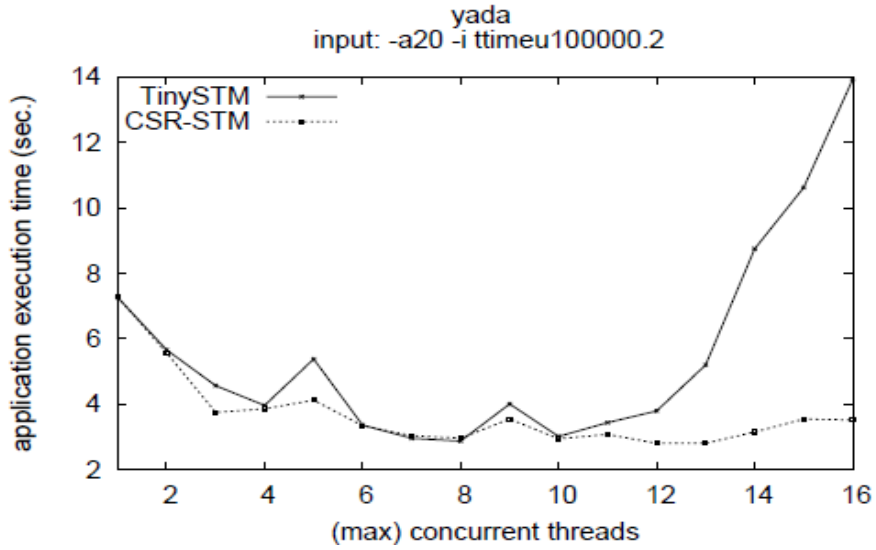
Hardware scalability model used for validation [14]:

$$C(k) = 1 + p \cdot (k-1) + q \cdot k \cdot (k-1)$$

Comparison Between CSR-STM and Tiny STM







Comparison with Other Approaches

• Analytical models of concurrency control protocols for transactional systems (e.g. [1,2,6,7,8]):

- strong assumptions on workload profile, operations' execution speed, data contention, etc → high prediction error with real applications

• Interpolation using different kind of functions (e.g. polynomial, rational, logarithmic functions) [8]:

- workload profile is not accounted → variation of the optimal concurrency level along the execution of the application can't be captured

• Machine learning-based approach (e.g. [5]) → low extrapolation capability (vs. the parametric performance model-based approach)

• Pro-active transaction scheduling schemes (e.g. [10,11,12])

- based on heuristic schemes → require evaluating suitable heuristics and tuning a set of thresholds depending on the application workload

Thank you

References (1/2)

[1] P. S. Yu, D. M. Dias, and S. S. Lavenberg. On the analytical modeling of database concurrency control. Journal of the ACM, pages 831–872, 1993.

[2] I. K. Ryu and A. Thomasian. Performance analysis of centralized databases with optimistic concurrency control. Performance Evaluation, 7(3):195–211, 1987.

[3] C. Cao Minh, J. Chung, C. Kozyrakis, and K. Olukotun. STAMP: Stanford transactional applications for multiprocessing. In Proc. 4th IEEE Int. Symposium on Workload Characterization, pages 35-46. 2008.

[4] P. Felber, C. Fetzer, and T. Riegel. Dynamic performance tuning of word-based software transactional memory. In Proc. 13th ACM Symposium on Principles and Practice of Parallel Programming, pages 237–246. 2008.

References (2/2)

[5] D. Rughetti, P. di Sanzo, B. Ciciani, and F. Quaglia. Machine learning-based self-adjusting concurrency in software transactional memory systems. In Proc. 20th Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecomm. Systems, pages 278–285. 2012.

[6] Y. C. Tay, N. Goodman, and R. Suri. Locking performance in centralized databases. ACM Transaction on Database Systems, pages 415–462, 1985.

[7] P. di Sanzo, B. Ciciani, F. Quaglia, and P. Romano. A performance model of multi-version concurrency control. In Proc. 16th Int. Symp. on Modeling, Analysis and Simulation
of Computer and Telecomm. Systems, pages 41–50. 2008.

[8] P. di Sanzo, R. Palmieri, B. Ciciani, F. Quaglia, and P. Romano. Analytical modeling of lock-based concurrency control with arbitrary transaction data access patterns. In Proc. 26th Int. Conf. on Performance Eng., pages 69–78. 2010.

[9] A. Dragojevi´c and R. Guerraoui. Predicting the scalability of an stm a pragmatic approach. In Proc. 5th ACM Workshop on Transactional Computing. 2010.

[10] A. Dragojevi´c, R. Guerraoui, A. V. Singh, and V. Singh. Preventing versus curing: avoiding conflicts in transactional memories. In Proc. 28th ACM Symposium on Principles of Distributed Computing, pages 7–16. 2009.

[11] M. Ansari, C. Kotselidis, K. Jarvis, M. Luj´an, C. Kirkham, and I. Watson. Advanced concurrency control for transactional memory using transaction commit rate. In Proc. 14thInt. Euro-Par Conference, pages 719–728. 2008.

[12] R. M. Yoo and H.-H. S. Lee. Adaptive transaction scheduling for transactional memory systems. In Proc. 20th Symp. On Parallelism in Algorithms and Archit., pages 169-178. 2008.

[13] N. J. Gunther. Guerrilla capacity planning - a tactical approach to planning for highly scalable applications and services. Springer, 2007.