Self-tuning Transactional Memory via Machine Learning and Analytical Modeling

Paolo Romano

INESC-ID
Instituto Superior Técnico, Lisbon University
Roadmap

• Background on Transactional Memory
  • alternative implementations

• TM performance tuning

• Gray box based self-tuning
  • Provisioning and optimization of a Distributed TM
  • Divide and conquer
  • Bootstrapping
  • Hybrid ensembling
The multi-core (r)evolution

Multi-cores are now ubiquitous

Concurrent programming is complex

Classic approach: Locking

- Hard to get right:
  - fine-grained locks
  - deadlocks
  - correctness

Transactional Memory abstraction

atomic {
    withdraw(acc1,val);
    deposit(acc2,val);
}

Programmer identifies atomic blocks
Runtime implements synchronization
(A very incomplete)
Historical perspective on TM

Intel’s Haswell CPU targets mainstream computing platforms:
• including desktops, servers, laptops, and tablets

Recently also IBM has integrated HTM supports in its high-end CPUs:
• BG/Q, zEC12, Power8
Transactional Memory: One abstraction, many implementations

- **Software (STM):**
  - instrumenting read and write accesses
    - PRO: flexibility
    - CON: instrumentation overheads

- **Hardware (HTM):**
  - extension of the cache consistency mechanism
    - PRO: no instrumentation overheads
    - CON: hw is inherently limited

- **Hybrid (HyTM):**
  - mix of the two worlds that tries to achieve the best of both

- **Distributed (DTM):**
  - natural extension of TM for distributed shared memory
    - PRO: fault-tolerance, potential for higher scalability
    - CON: synchronization costs are amplified
Software TM

Source program

```c
int i = 0
...
atomic {
  i++
}
```

Compiled program

```c
int i = 0
...
TM.begin-tx()
int tmp = TM.read(&i)
tmp++
TM.write(&i, tmp)
TM.end-tx()
```

- Non-negligible instrumentation overheads
- Highly flexible:
  - Avoid inherent restrictions of hardware implementations
  - Over 10 years of research on STM
    - highly optimized prototypes and designs
Transactional Memory:
One abstraction, many implementations

• Software (STM):
  • instrumenting read and write accesses
    • PRO: flexibility
    • CON: instrumentation overheads

• Hardware (HTM):
  • extension of the cache consistency mechanism
    • PRO: no instrumentation overheads
    • CON: hw is inherently limited

• Hybrid (HyTM)
  • mix of the two worlds that tries to achieve the best of both

• Distributed (DTM)
  • natural extension of TM for distributed shared memory
    • PRO: fault-tolerance, potential for higher scalability
    • CON: synchronization costs are amplified
HTM: Intel Transactional Synchronization Extensions (TSX)

```
xbegin
read x: 0  // Set bit read on x cache line
write y = 1 // Buffer write in L1 cache
xend // Atomically clean bits and publish

write y = 2
```

```
xbegin
read y: 1
write y = 2
xabort
```

Invalidation of the tx read by the snooped write.

L1 Cache  L2 Cache  L3 Cache
CPU 1  CPU 2  L1 Cache
TSX: on

Memory Bus
Restrictions of TSX

No progress guarantees:

• A transaction may always abort

...due to a number of reasons:

• Forbidden instructions
• Capacity of caches
• Faults and signals
• Contending transactions, aborting each other

TSX alone is not enough

Needs software fall-back:

• global lock $\rightarrow$ standard HTM
• STM $\rightarrow$ hybrid TM
Transactional Memory: One abstraction, many implementations

- **Software (STM):**
  - instrumenting read and write accesses
    - PRO: flexibility
    - CON: instrumentation overheads

- **Hardware (HTM):**
  - extension of the cache consistency mechanism
    - PRO: no instrumentation overheads
    - CON: hw is inherently limited

- **Hybrid (HyTM)**
  - mix of the two worlds that tries to achieve the best of both

- **Distributed (DTM)**
  - natural extension of TM for distributed shared memory
    - PRO: fault-tolerance, potential for higher scalability
    - CON: synchronization costs are amplified
Distributed Transactional Memory

- Extends the reach of TM abstraction to distributed applications
- Enhanced scalability, high-availability and fault-tolerance
- Attractive paradigm for the cloud
At the convergence of two areas

**Distributed Shared Memory**

Transactions allow to:
1. Deal with remote data races
2. Boost performance by batching remote synchronizations during commit phase

**Distributed Databases**

- Natural source of inspiration for DSTMs...
- but DSTMs have unique requirements, e.g.:
  - >70% txs are 100x shorter in DSTM

**Distributed Transactional Memory**
Roadmap

• Background on Transactional Memory
  • alternative implementations

• TM performance tuning

• Gray box based self-tuning
  • Provisioning and optimization of a Distributed TM
  • Divide and conquer
  • Bootstrapping
  • Hybrid ensembling
TM performance tuning

- TM abstraction allows for encapsulating a vast range of alternative implementation strategies
  - no one size fits all solution [SPAA08, PACT14]

- Each implementation comes with various tuning-knobs:
  - number of retries in HTM [ICAC14]
  - granularity of locks in STM [PPoPP08]

- Parallelism degree:
  - how many threads should be concurrently active? [EuroPar14]

- Thread mapping:
  - on which cores should the active threads be executed? [JPDC14]
TM tuning: no one size fits all

(a) Throughput/Joule on a single-chip 8-core CPU - Machine A

(b) Throughput on a multi-chip 32-core CPU - Machine B

<table>
<thead>
<tr>
<th>Machine ID</th>
<th>Processor / Number of cores / RAM</th>
<th>HTM</th>
<th>RAPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine A</td>
<td>1 Intel Haswell Xeon E3-1275 3.5GHz / 4 (8 hyper-threads) / 32 GB</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Machine B</td>
<td>4 AMD Opteron 6172 2.1 Ghz / 48 / 32 GB</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
DTM performance tuning

- Support both scale up and scale out [TAAS14]
  - how many machines should my DTM be provisioned with?
  - how many threads should be active on each machine?

- Communication latencies play a critical factor
  - select the distributed coordination protocol that maximizes efficiency [DSN13]
  - dynamically tune parameters (e.g., batching) of the Group Communication System to enhance efficiency [ICPE15]
  - where should data and code be placed to maximize locality? [ICAC13]

- Cost of exploration can be much higher [Netys13]:
  - launching a new VM is not as simple as spawning a new thread:
    - latency for VM activation, system reconfiguration, state transfer
    - economical cost for VM activation in the cloud
Performance of Distributed TM

- Heterogeneous, nonlinear scalability trends!
DTM: Factors limiting scalability

Network latency in commit phase

Aborted transactions because of conflicts
Roadmap

• Background on Transactional Memory
  • alternative implementations

• TM performance tuning

• Gray box based self-tuning
  • Provisioning and optimization of a Distributed TM
  • Divide and conquer
  • Bootstrapping
  • Hybrid ensembling
Based on the following papers


Approaches to Performance Modelling

White Box

Black Box
White box modelling

• Exploit knowledge on internal system dynamics
  ✦ model dynamics analytically or via simulation

PROS

• Good accuracy **on average**

• Minimal or no learning phase

CONS

• Simplifying assumptions
  ➔ low accuracy when these assumptions do not hold

• Knowledge of system internals often unavailable
Related Work

Machine Learning

Goal

Observe

\[ x = (x_1, \ldots, x_n), \]

\[ y = (y_1, \ldots, y_n) \]

Infer

\[ y' = f(x') \]

Machine Learner

Training input

\( \{x_i\} \)

Training output

\( \{y_i = f(x_i)\} \)

Statistical Model

\( y = f'(x) \)

Query \( x' \)

PROS

- High accuracy in areas already observed (interpolation)
- Do not require knowledge on system’s internals

CONS

- Poor accuracy in non-observed areas (extrapolation)
- Curse of dimensionality
  - Extensive training phases

Black box modelling

Diego Didona (2013)
Key Observation & Questions

Pros of white-box are cons of black-box & vicev.

Can we achieve the best of the two worlds?

How can black and white box modelling be reconciled?
Gray box modeling

• Combine WB and BB modeling

• Enhance **robustness**
  – Lower training time thx to WBM
  – Incremental learning thx to BBM

• Will present three methodologies:

  ![Diagram showing three methodologies]

  - Divide and conquer
  - Bootstrapping
  - Hybrid ensembling
Gray box modeling

- Will present three methodologies:
  - Divide and conquer
  - Bootstrapping
  - Hybrid ensembling
Divide and conquer

💡 Modular approach
  – WBM of what is observable/easy to model
  – BBM of what is un-observable or too complex

• Reconcile their output in a single function

👍 Higher accuracy in extrapolation thx to WBM
👍 Apply BBM only to sub-problem
  – Less features, lower training time
Case study: Infinispan

• Distributed in-memory key-value store:
  – Nodes maintain elements of a dataset
    • Full vs partial replication (# copies per item)
  – Transactional --ACI(D)-- manipulation of data
    • Concurrency control scheme (enforce isolation)
    • Replication protocol (disseminate modifications)
DTM optimization in the Cloud

- Important to model network-bound ops but...

😊 Cloud hides detail about network 😞
  - No topology info
  - No service demand info
  - Additional overhead of virtualization layer

💡 BBM of network-bound ops performance
  - Train ML on the target platform
TAS/PROMPT [TAAS14,Mascots14]

• Analytical modeling (queuing theory based)
  – Concurrency control scheme
    • E.g., encounter time vs commit time locking
  – Replication protocol
    • E.g., PB vs 2PC
  – Replication scheme
    • Partial vs full
  – CPU

• Machine Learning
  – Network bound op (prepare, remote gets)
  – Decision tree regressor
Analytical model in TAS/PROMPT

• Concurrency control scheme (lock-based)
  – A lock is a M/G/1 server
  – Conflict prob = utilization of the server

• Replication protocol
  – multi-master/Two-phase Commit based \(\rightarrow\) one model
  – single-master/primary-backup \(\rightarrow\) two models

• Replication scheme
  – Probability of accessing remote data
  – # nodes involved in commit
Machine Learning in TAS/PROMPT

• Decision tree regressor
• Operation-specific models
  – Latency during prepare
  – Latency to retrieve remote data
• Input
  – Operations rate (prepare, commit, remote get...)
  – Size of messages
  – # nodes involved in commit
ML accuracy for network bound ops

Seamlessly portable across infrastructures
– Here, private cloud and Amazon EC2
AM and ML coupling

At training time, all features are monitorable

At query time they are NOT!

EXAMPLE

• Current config: 5 nodes, full replication
  – Contact all 5 nodes at commit

• Query config: 10 nodes, partial replication
  – How many contacted nodes at commit??
Model resolution

💡 AM can provide (estimates of) missing input
  
  • Iterative coupling scheme

ML takes some input parameters from AM

AM takes latencies forecast by ML as input parameter
Model’s accuracy

TOP: primary-backup. BOTTOM: multi-master (2PC-based)
Comparison with Pure ML, I

- YCSB (transactified) workloads while varying
  - # operations/tx
  - Transactional mix
  - Scale
  - Replication degree
ML trained with TPCC-R and queried for TPCC-W
Pure ML blunders when faced with new workloads
Gray box modeling

• Will present three methodologies:

  Divide and conquer
  Bootstrapping
  Hybrid ensembling
Bootstrapping

💡 Obtain zero-training-time ML via initial AM
1. Initial (synthetic) training set of ML from AM
2. Retrain periodically with “real” samples
How many synthetic samples?

- **Important tradeoff**
  - Higher # $\rightarrow$ lower fitting error over the AM output
  - Lower # $\rightarrow$ higher density of real samples in dataset
How to update the synthetic training set?

- Merge: simply add real samples to synthetic set
- Replace only the nearest neighbor (RNN)
- Replace neighbors in a given region (RNR)
  - Two variants
Real vs AM function
• Assuming enough point to perfectly learn AM
Merge

- Add real samples to synthetic
• Problem: same/near samples have diff. output
Replace Nearest Neighbor (RNN)

• Remove nearest neighbor
Replace Nearest Neighbor (RNN)

- Preserve distribution...
Replace Nearest Neighbor (RNN)

- ... but may induce alternating outputs
Replace Nearest Region (RNR)

- Add real and **remove** synth. samples in a radius
Replace Nearest Region (RNR)

- $R = \text{radius defining neighborhood}$
Replace Nearest Region (RNR)

• $R =$ radius defining neighborhood
Replace Nearest Region (RNR)

- Skew samples’ distribution
Replace Nearest Region 2 (RNR2)

- **Replace** all synthetic samples in a radius R
Replace Nearest Region 2 (RNR2)

- Maintain distribution, piecewise approximation
Weighting

• Give more relevance to some samples

👍 Fit better the model around real samples
  – “Trust” real samples more than synthetic ones
  – Useful especially in Merge

👎 Too high can cause over-fitting!
  – Learner fails to generalize
Evaluation

• Case studies
  – Response time in Total Order Broadcast (TOB)
    • building block at the basis of many DTM
    • 2-dimensional yet highly nonlinear perf. Function
  – Throughput in Distributed TM (Infinispan)
    • 7-dimensional performance function
This section evaluates the alternative algorithms for the updating of the knowledge base, that is, it first assesses the sensitivity of each algorithm to its key parameters and finally compares their accuracy assuming an optimal tuning of such parameters. Such model may be unable to properly approximate the performance function of strongly non-linear behaviors.

Figure 5.5: Impact of the weight parameter for the Merge updating policy, using 1K and 10K synthetic samples.

Weighting

TOB
(10K synthetic samples)

DTM
(10K synthetic samples)
• In both considered case studies, simplicity pays off:
  – the Merge policy performs analogously to RNR2
  – ...but, unlike RNR2, Merge is parameter-free
Visualizing the correction

BASE MODEL

BOOTSTRAPPED ML (70% TS)

PURE ML (70% TS)
Gray box modeling

- Will present three methodologies:
  - Hybrid ensembling
  - Divide and conquer
  - Bootstrapping
  - Hybrid boosting
  - Hybrid KNN
  - Probing
Hybrid Boosting

💡 Learning the error of a model on a function may be simpler than learning the function itself

- Chain composed by AM + cascade of ML
- $ML_1$ trained over residual error of AM
- $ML_i, \ i > 1$ trained over residual error of $ML_{i-1}$
Training and Querying Hyboost

Original training set

\(<x_1, y_1>\)
\(<x_2, y_2>\)
\(.\)
\(.\)
\(<x_n, y_n>\)
Training and Querying Hyboost

Original training set

\[ \langle x_1, y_1 \rangle, \langle x_2, y_2 \rangle, \ldots, \langle x_n, y_n \rangle \]

Training

AM

Residual error of AM

\[ \langle x_1, y_1 - \text{AM}(x_1) \rangle, \langle x_2, y_2 - \text{AM}(x_2) \rangle, \ldots, \langle x_n, y_n - \text{AM}(x_n) \rangle \]
Training and Querying Hyboost

Training

Original training set

\(<x_1, y_1>\),
\(<x_2, y_2>\),
\ldots
\,<x_n, y_n>\)

AM

Residual error of AM

\(<x_1, y_1 - AM(x_1)>\),
\,<x_2, y_2 - AM(x_2)>\),
\ldots
\,<x_n, y_n - AM(x_n)>\)

ML_1

Residual error of ML_1

\,<x_1, y_1 - ML_1(x_1)>\),
\,<x_2, y_2 - ML_1(x_2)>\),
\ldots
\,<x_n, y_n - ML_1(x_n)>\)
Training and Querying Hyboost

Training

Original training set

$\langle x_1, y_1 \rangle$
$\langle x_2, y_2 \rangle$
$\vdots$
$\langle x_n, y_n \rangle$

AM

Residual error of AM

$\langle x_1, y_1 - AM(x_1) \rangle$
$\langle x_2, y_2 - AM(x_2) \rangle$
$\vdots$
$\langle x_n, y_n - AM(x_n) \rangle$

ML$_1$

Residual error of ML$_1$

$\langle x_1, y_1 - ML_1(x_1) \rangle$
$\langle x_2, y_2 - ML_1(x_2) \rangle$
$\vdots$
$\langle x_n, y_n - ML_1(x_n) \rangle$

Query

$F(x) = AM(x) + ML_1(x) + \ldots + ML_m(x)$
• Will present three methodologies:

Gray box modeling

- Divide and conquer
- Bootstrapping
- Hybrid boosting
- Probing

Hybrid ensembling

Hybrid KNN
Hybrid KNN

💡 Predict performance of $x$ with model that is supposed to be the most accurate for it

• Split training set $D$ into $D'$, $D''$

• Train $ML_1 \ldots ML_N$ on $D'$
  – ML can differ in nature, parameters, training set...

• For a query sample $z$
  – Pick the $K$ training samples in $D''$ closer to $z$
  – Find the model with lowest error on the $K$ samples
  – Use such model to predict $f(x)$
KNN Training and Querying

TRAINING SET

ML_1 \quad \ldots \quad ML_n

AM
KNN Training and Querying

ML TRAINING SET

KNN TRAINING SET

ML₁ ... MLₙ

AM
KNN Training and Querying

ML TRAINING SET

KNN TRAINING SET

KNN, CUT-OFF C

ML\textsubscript{1}

...\quad ML\textsubscript{n}

AM

QUERYING

NEAREST NEIGHBORS

X
KNN Training and Querying

ML TRAINING SET

KNN TRAINING SET

KNN, CUT-OFF C

QUERYING

ML₁  ...  MLₙ

EVALUATE

NEAREST NEIGHBORS

ACCURACY

AM

X
KNN Training and Querying

ML TRAINING SET

KNN TRAINING SET

KNN, CUT-OFF C

ML \_1 \ldots ML \_n

EVALUATE ACCURACY

NEAREST NEIGHBORS

MOST ACCURATE MODEL

AM

QUERYING

X
KNN Training and Querying

ML TRAINING SET

KNN TRAINING SET

KNN, CUT-OFF C

X

ML\_1

...\n
ML\_n

ML\_1, ... ML\_n

EVALUATE

ACCURACY

NEAREST NEIGHBORS

MOST ACCURATE MODEL

F(X)

QUERYING

F(X) = \( \text{most accurate model} \)
Gray box modeling

- Will present three methodologies:
  - Divide and conquer
  - Bootstrapping
  - Hybrid ensembling
  - Hybrid boosting
  - Probing
  - Hybrid KNN
Probing

💡 Build a ML model as specialized as possible
  – Use AM where it is accurate
  – Train ML only where AM fails

⚠️ Differences w.r.t. KNN
  – Training: in KNN, ML is trained on all samples:
    • Here, ML trained on samples for which AM is inaccurate
  – Querying: In KNN, voting decides on ML vs AM
    • Here, binary classifier predicts when the AM is inaccurate
Probing training and querying

Original training set

ML training set

Classifier training set

AM
Probing training and querying

TRAINING

ML training set

Original training set

Classifier training set

<\mathbf{x}_1, y_1>

AM

ERROR < CUT-OFF?
Probing training and querying

TRAINING

ML training set

Original training set

Classifier training set

<\textbf{x}_1, \textbf{y}_1>

<\textbf{x}_1, \textbf{AM}>

AM

ERROR < CUT-OFF?

YES
Probing training and querying

TRAINING

ML training set

<\(x_2, y_2\)>

Original training set

<\(x_2, y_2\)>

Classifier training set

<\(x_1, AM\)>

<\(x_2, ML\)>

AM

NO

ERROR < CUT-OFF?

NO

NO
Probing training and querying

\[ F(x) = \begin{cases} 
AM(x) & \text{if } \text{Classify}(x) = \text{AM} \\
ML(x) & \text{otherwise} 
\end{cases} \]
Evaluation

• Sensitivity to meta-parameters
  – Hyboost
    • Size of the chain
  – Hybrid KNN
    • Proximity cut-off
  – Probing
    • Minimum AM’s accuracy cut-off

• Comparison among the techniques
• Chain composed by AM + Decision Tree
• Longer chains yielded negligible improvements in the considered case studies
Tuning of hyper-parameters matters

- **Comparison**
  - Pure AM, Pure ML (Cubist, Decision tree regressor) vs
  - Probing (AM + Cubist)

- **Analogous considerations hold for KNN**
No free lunch theorem strikes again

- No one-size-fits-all hybrid model exists
- Tackle choice of best hybrid model via cross-validation

Figure 5.14: Comparing the performance of the 4 proposed gray box techniques.

5.4.6 Comparison among the approaches
This section concludes the experimental evaluation and is dedicated to comparing the accuracy achieved by the four proposed hybrid ensemble techniques in the two considered case studies. In particular, the comparison is performed assuming a proper tuning of the internal parameters of the compared ensemble algorithms. Specifically, the reported data are obtained using 10-fold cross validation to determine appropriate values for the internal parameters of the compared ensemble algorithms.

As already hinted in Section 5.4.1.3, identifying the best gray box model and corresponding parameterization given some training data is a problem that falls beyond the scope of the proposed Hybrid Ensemble techniques: it is, indeed, a common trait shared with pure black box modeling techniques. Therefore, it can be tackled by means of standard techniques developed for the selection and tuning of Machine Learning algorithms, such as Bayesian Optimization or grid/random search (Bergstra et al., 2011).

The following evaluation aims at showing how the characteristics of the target performance function and of a hybrid predictor affect accuracy in the most favorable case, i.e., excluding the cases in which a given predictor performs poorly only because of a correspondingly poor setting of its internal parameters.
AM error vs optimal technique

- Error distribution of the base AM is key
- Hy-boost performed the best for DTM
  - Smooth error function is easy to learn
- Not the case for TOB
  - Highly localized errors better tackled via probing
Concluding remarks: TM and Self-tuning

• Transactional memory is an attractive alternative to lock-based synchronization:
  – hides complexity behind intuitive abstraction
  – relevance amplified by integration with GCC, commodity (Intel’s) and HPC (IBM’s) CPUs

• Performance of TM is strongly affected by:
  – workload characteristics
  – choice of the TM implementation
  – plethora of implementation-dependent parameters

• Self-tuning is critical to ensure efficiency!
Concluding remarks: Which modeling methodology?

💡 White and black box models can be effectively used in synergy
  - Increased predictive power via analytical models
  - Incremental learning capabilities via black box models

Presented three gray box methodologies:
  - Divide and conquer, Bootstrapping, Hybrid ensembling
  - Design, implementation and application to (D)TM

❗ Careful choice of technique and parameters
  🌡️ Use standard techniques for hyper-parameters opt.
Open questions

• Any other way of hybridizing Black and White modelling?

• Can we further combine them?
  • e.g. use a bootstrapped model in an ensemble?

• Can we infer the best gray box technique by analyzing the error function of the AM model?
References

- [SPAA08] Torvald Riegel, Christof Fetzer, Pascal Felber, Automatic data partitioning in software transactional memories. SPAA 2008: 152-159
- [PPoPP08] Pascal Felber, Christof Fetzer, Torvald Riegel, Dynamic performance tuning of word-based software transactional memory. PPOPP 2008: 237-246
THANK YOU

Questions?
romano@inesc-id.pt
www.gsd.inesc-id.pt/~romanop