A recommendation system approach to the tuning of Transactional Memory

André Alves Rogério Santos
andrealvesrogeriosantos@ist.utl.pt

Instituto Superior Técnico, Lisboa, Portugal

May 2017

Abstract

Transactional Memory (TM) is a promising approach that simplifies parallel programming. However, in the broad spectrum of available TM implementations, there exists no one size fits all solution that can provide optimal performance across all possible workloads. This has motivated an intense research, over the last years, on the design of self-tuning solutions aimed at transparently adapting the choice and configuration of the TM run-time system. This report focuses on advancing a recent, state of the art solution in the area of TM self-tuning, called ProteusTM. ProteusTM builds on recommendation systems and Bayesian analysis techniques in order to automate the tuning process of a TM system over a multi-dimensional configuration space — a unique feature in the literature on TM self-tuning. In particular, this report investigates two key research questions: i) how to extend ProteusTM to support sparse training sets, and ii) to what extent can the inclusion of workload characteristics (e.g., abort rate) enhance the accuracy achieved by ProteusTM.
Keywords: parallel programming, transactional memory, self-tuning, ProteusTM, recommender systems

1. Introduction

Multi-core architectures are nowadays ubiquitous, bringing parallel programming to the forefront of software development, with the objective of achieving better performance results.

Transactional Memory (TM) [1] is a promising approach to solve the problems of concurrent programming, based on a key concept from the database community, namely transactions. TM offers a powerful abstraction to programmers, reducing the complexity of building parallel programming applications. With the TM abstraction programmers need only to specify *which* region of code to run atomically without worrying about *how* concurrent accesses should be synchronized (in contrast with locks) to guarantee correctness.

Over the years, several works have provided evidence of the effectiveness of TM in simplifying the development of parallel programs and delivering good performance [5, 11, 16]. However, in order to pave the ground for the adoption of TM as a mainstream paradigm for parallel programming, there is one crucial issue to address. Despite the numerous existing implementations, either in software or hardware, there exists no single solution that can provide optimal performance across all possible workloads [6, 28, 8]. Research has provided evidence that the choice of the best TM implementation and of its internal parameters (i.e., number of threads) is affected, in complex ways, by a number of workload characteristics [6, 9, 8]. In addition, performance can be affected by several factors, for instance programs inputs, phases of program execution, as well as the architectural aspects of the underlying hardware. Further, as heterogeneity of hardware/software architecture keeps on increasing, it appears quite unlikely that an "universal" solution will be identified in the near future.

Self-Tuning reveals as an appealing solution to cope with heterogeneity in workloads and TM optimization, removing the burden of the programmer to identify the optimal configuration. Existing self-tuning techniques for TM systems [8, 6, 28] rely on modelling and forecasting techniques to optimize TM performance, , i.e., to identify the optimal configuration of one or more parameters controlling the behaviour of a TM algorithm based on the workload generated by some target application. Some example of black-box modelling techniques used in self-tuning systems for TM include neural networks [2], decision trees [20] and collaborative filtering [17].

Most of existing works using self-tuning to optimize TM performance aim to adapt dynamically either: (i) internal TM parameters, for instance

number of retries when a transaction aborts [8]; (*ii*) the concurrency level of a TM, adapting the number of active threads [24, 7]; (*iii*) the choice of the TM algorithm to employ [28]. However, these solutions optimize the TM run-time along a single dimension, which is clearly unsatisfactory given the multitude of configuration choices/alternative implementations that exist.

To the best of my knowledge, the first and only multi-dimensional self-tuning solution for TM is *ProteusTM* [6]. For this reason, we have chosen ProteusTM as the main focus of this study.

ProteusTM leverages on Collaborative Filtering and Baeysian Optimization to identify the configuration that optimizes a given Key Performance Index (KPI) (e.g., throughput, execution time).

The goal of this dissertation is to investigate two research questions originated by ProteusTM:

1. One of the main problems of ProteusTM is that it requires a key pre-processing step, namely Rating Distilation (RD), which assumes the availability of full information regarding the performance of *every* application (included in the training set) across *every* available TM configurations. This is quite a relevant limitation, given that even in a small scale system the number of available TM configurations are on the order of hundreds and that the configuration space grow exponentially with its dimensionality (the well-known curse of dimensionality [3])

   The first research question addressed, deals precisely with this limitation of ProteusTM, proposing and systematically evaluating techniques aimed at supporting the use of sparse training information.

2. ProteusTM's training phase relies solely on the knowledge of the target KPI achieved by a workload when deployed over different set of configurations. In other words, it does not exploit any information regarding intrinsic workload characteristics (e.g., abort rate or transaction duration). Hence, a natural question, which is addressed in this dissertation, is to what extent can the accuracy of ProteusTM be improved by incorporating information on workload characteristics in its knowledge base.

   We provide an answer to this question by considering different learners and normalization techniques.

This document is structured as follows. Chapter 2 provides a background on TM and Collaborative Filtering (CF) techniques. Chapter 3 describes ProteusTM in detail, and highlights the research directions we intend to study in this report. Chapter 4 addresses the problem of how to support sparse training sets in ProteusTM. Chapter 5 studies the problem of incorporating workload information in ProteusTM's knowledge base. Finally, Chapter 6 concludes the report.

2. Background

This section provides some background about TM and an overview CF techniques from the Recommendation Systems (RS) literature.

2.1. Transactional Memory

TM is a concurrency-control mechanism that reduces the complexity of building parallel programming applications, relying on a powerful abstraction used for decades in the database community, namely transactions [1]. In practice, a TM system runs several transactions in parallel, and is up to the TM library to enforce isolation and atomicity via some concurrency-control algorithm.

TM systems allow transactions to run in parallel, by relying on the abstraction of atomic blocks to determine which portions of the code to run as atomic code blocks. Consequently the problem of regulating concurrency is shifted from the programmers to the TM designers. Over the years many different TM designs have been proposed to tackle this issue. We discuss them in the next paragraph which have to consider the key mechanisms of TM design such as data versioning and conflict detection.

2.1.1 TM implementations:

The TM abstraction has been implemented either in software (STM) [5], hardware (HTM) [8] and hybrid-approaches [4]. Software Transactional Memory (STM) implements the TM abstraction entirely in software and needs costly software instrumentation of reads and writes memory accesses to trace conflicts between concurrent transactions. HTM, instead leverages on the processor's cache coherence mechanism to track memory accesses. On the one hand this leads to lower overhead and, thus, potentially better performance. On the other hand it requires that the memory accesses performed by a transaction can be entirely stored in the cache hierarchy. Hybrid TMs combine the two approaches: transactions are executed in HTM whenever possible, and in STM when necessary.

Despite the variety of TM implementations [11, 5, 16, 21], existing literature in the area seems to have reached consensus on the fact that there is no "one-size-fit-all" solution that can provide optimal performance across all possible workloads. In fact the best TM for a given workload can deliver performance that are orders of magnitude lower than the optimal for another workload [6].

Self tuning represents a very appealing solution

to tackle this issue. A step towards this direction is represented by ProteusTM. Next, we describe CF, namely the technique used by ProteusTM to predict the quality of different configurations depending on the workload.

2.2. Collaborative Filtering

CF is one of the most popular techniques in the literature of RS. The basis of CF is that if user X and Y rate the same items similarly, they will rate or recommend other items similarly [27].

In order to infer a rating between a $\langle user, item \rangle$ pair, CF exploits the preferences expressed by other users or the relationships between items. CF systems use a database of preferences in order to create an Utility Matrix (UM): the rows represent the users and the columns the various items. The elements of the UM are the ratings the user attributes to an item, normally in very homogeneous scales (e.g. 0-5 stars). In order to give recommendations CF algorithms fill the missing ratings of the UM which are typically very sparse. Two of the most prominent approaches to predict missing ratings are: K Nearest Neighbours (KNN) and Matrix Factorization (MF).

KNN uses a *similitarity function* that identifies similarities between users or items [10]. To compute the recommendations if first chooses the neighbourhood of K elements. Then the rating predictions are normally obtained by computing the weighted average of the neighbouring user's/item's ratings using the similarity as the weights.

MF maps both users and items to a joint latent vector space of dimensionality [17], where the user-item interactions are modelled as inner products in that space. Intuitively, borrowing from the movie recommendation scenario, MF tries to understand how much a movie belongs to hidden categories such as drama or action, and how much a user likes those genres. Single-Value Decomposition (SVD) is one of the commonly adopted method to identify and extract latent factors, which aims to reconstruct a *full* matrix starting from the sparse initial UM, M. It infers two matrices P and Q, which represent, respectively, users and items in the aforementioned f-dimensional space, and computes $R = Q^T P \approx M$, which contains the missing values of M .

3. Study on ProteusTM

To the best of my knowledge, ProteusTM [6] is the only self tuning solution for TM systems that supports a multi-dimensional optimization scheme.

At its core ProteusTM has two main components *PolyTM* and *RecTM* (see Figure 1). PolyTM consists of a Polymorphic TM library, with several TM implementations (HTM, STM, HyTM) and allows the reconfiguration of TM along the following dimensions: (i) switch between different TM implementations (TinySTM to NOrec for example); (ii) reconfigure internal parameters of a TM; (iii) adapt the number of threads concurrently generating transactions.

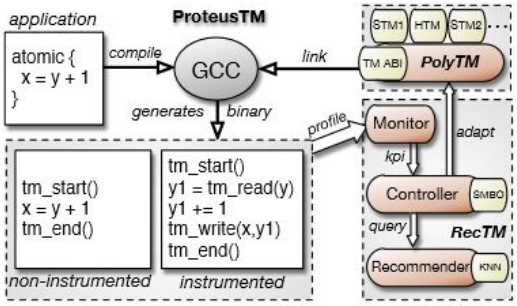


Figure 1: ProteusTM [6] architecture.

RecTM, the main component studied for this report, is responsible for identifying the best configuration of PolyTM for the current workload. Briefly, ProteusTM applies CF to the problem of identifying the best TM configuration that maximizes the KPI for some workload.

3.1. RecTM

In Figure 1 note that RecTM, the component responsible for predicting the best TM configuration, is composed by 3 sub-modules [6]:

1. **Recommender**, the recommendation system that acts as the predictor and supports different CF algorithms.
2. **Controller**, which selects the configuration to be used and triggers the adaptation in PolyTM, by queriyng the Recommender with the values obtained from the Monitor.
3. **Monitor**, which is responsible for gathering the target KPI to give feedback to the Controller about the quality of the current configuration. Also, it detects changes in the workloads with the objective of triggering a new optimization phase on the Controller.

As mentioned, RecTM casts the performance prediction task to a recommendation problem. In ProteusTM the UM associates each row to a different workload, and each column to a different TM configuration. Exploiting CF-based recommendation systems for the self-tuning of TM raises a non-trivial problem, which is discussed next.

3.1.1 The Rating Heterogeneity Problem

CF algorithms assume the use of homogeneous rating scales, e.g., from 0 to 5 stars. Conversely, performance metrics of TMs (such as throughput, execution time, abort rate) typically span across very

Machine ID	TM Backend	# threads	HTM Abort Budget	HTM Capacity Abort Policy
Machine A	STMs and TSX [67]	1,2,3,4, 5,6,7,8	1,2,4, 8,16,20	Set budget to 0; decrease budget by 1; halve budget
Machine B	STMs	1,2,4,6, 8,16,32,48	N/A	N/A

Figure 2: Parameters tuned by ProteusTM. STMs are TinySTM [11], SwissTM , NORec [5] and TL2

heterogeneous scales, depending on the applications characteristics (e.g., long vs short transactions) and hardware characteristics (e.g., CPU clock speed).

A solution to the problem of heterogeneity is to normalize the whole UM. Normalization in this scenarios is not trivial because the best and worst configuration for a given workload (and their corresponding performance) are not known a priori. Ideally, an efficient normalization should be able to transform the entries so that similarities can be mined and enable the use of CF techniques.

To further motivate the use of normalization we present a study on the performance of ProteusTM predictor capabilities, using different methods of normalization.

The purpose of this study is to assess the effect of different normalization strategies on the quality of the recommendation made by ProteusTM. The different types of normalization used were:

(i) NONE: No normalization, the CF algorithm is applied in the raw UM; *(ii)* MAX: normalization with respect to the max in the training set; *(iii)* WRT-BEST/IDEAL: an ideal normalization technique that assumes to know a priori the absolute value of the KPI in the optimal configuration for each workload and is then used as normalization factor. *(iv)* Rating Distillation (RD): a normalization procedure proposed in the ProteusTM work [6] and described more in detail in the following subsection.

The performance of ProteusTM will be evaluated using two accuracy metrics: Mean Average Percentage Error (MAPE) and Mean Distance From Optimum (MDFO).

We note $r_{u,i}$ represents the real value of the target KPI for workload u when running i as configuration, $\hat{r_{u,i}}$ the corresponding prediction of the Recommender, and S the set of testing $\langle u, i \rangle$. The MAPE is defined as:$\sum_{\langle u,i \rangle \in S} |r_{u,i} - \hat{r_{u,i}}|/r_{u,i}$.

Note i^*_u is the optimal configuration for workload u, and $\hat{i^*_u}$ the best configuration found be the Recommender. The MDFO is identified by: $\sum_{\langle u,i \rangle \in S} |r_{u,i^*_u} - r_{u,\hat{i^*_u}}|/r_{u,i^*_u}$.

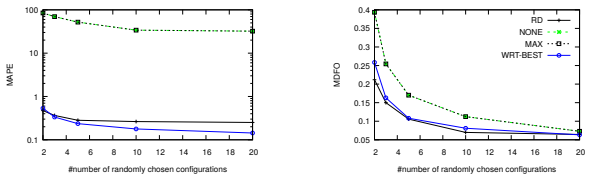
The MAPE reflects how well the CF learner predicts performance for an application, while MDFO captures the quality of the final recommendations.

In order to build the experimental test-bed used in all the experiments of this report, ProteusTM was deployed in two machines with different characteristics with a wide variety of TM applications, and tuning parameters depicted in Figure 2. In these two machines, over 300 workloads were deployed which are representative of heterogeneous applications such as: STAMP [19], Data Structures, STMBench7 [12], TPC-C and Memcached [14]. The experience data set was built by collecting over 5 runs, the KPI (e.g., throughput and execution time) in a real-time trace driven evaluation of over 300 workloads and 160 TM configurations.

The KPI used for this study was the execution time.

The evaluation's learner was KNN with cosine similarity and two neighbours, depicted in Figure 3 trained with a random sub-set of the original data set split into 30% of training set and the remaining 70% the corresponding test set. The results presented are an average of 10 runs, for 10 different sub-sets of the original data set and for a different number of configurations with known performance for a given workload (chosen at random). The training set is used to instantiate the predictive model and where the normalization techniques are applied. The test-set has no intersection with the training set and provides each workload to ProteusTM, so that this can predict the values not present in the sampling. To simulate sampling for the performance of an workload for a given configuration, we use the corresponding value from the test-set and add this value to the UM of the Recommender.

To ensure fairness, the study was conducted with the same training sets and the same initial configurations were provided for the different normalizations. Figure 3a represents how far in average are the predictions from the actual rating, while Figure 3b quantifies the quality of the final recommendation, i.e., how far is the recommended configuration from the actual optimum.



(a) MAPE, KNN Cosine (b) MDFO, KNN Cosine

Figure 3: Normality study using ProteusTM for a 30% training set and 70% test set

The results depicted in Figure 3 show that the predictive accuracy of CF algorithms is strongly affected by the choice of the normalization procedure. We can verify that applying no normalization or using the normalization with respect to the max, the predictor performs very poorly in terms of overall predictions (Fig. 3a) and when finding the optimal

configuration (Fig. 3b). While using an ideal solution or RD the recommendations are really close to the optimum configuration and to the overall absolute values of the predictions.

3.1.2 Normalization in the Recommender

An ideal normalization cannot be used in ProteusTM since it would require knowing a priori the KPI of the optimal configuration for an unknown workload.

RD, the normalization techniques proposed in the ProteusTM's work [6], aims at approximating the ideal solution described in the previous sub section by ensuring that for any workload w: *(i)* of two configurations c_i and c_j, namely kpi_w, c_i and kpi_w, cj, the ratio is preserved in the rating space, i.e., $\frac{kpi_{w,c_i}}{kpi_{w,c_j}} = \frac{r_{w,c_i}}{r_{w,c_j}}$ where r_w, ci and r_c, c_j represent the ratings attributed, respectively, to configurations ci and cj for workload w; *(i)* the ratings of the various configurations of a workload w are distributed in the range [0, M_w] so as to minimize the the index of dispersion of M_w : $D(M_w) = \frac{var(M_w)}{mean(M_w)}$.

Algorithm 1 Rating Distillation algorithm [6].

1: **for** $C_i \in C_1 \dots C_K$ **do**
2: Normalize Matrix KPI w.r.t. C_i
3: Collect the vector M_w with the max values per row
4: Compute $mean_i(M_w)$ and $var_i(M_w)$
5: **end for**
6: Return C* = $argmin_{i \in 1 \dots M}\ var_i(M_w)/mean_i(M_w)$

Rating Distillation (RD) used by the Recommender is depicted in Algorithm 1. RD is a normalization technique that assumes the availability, for each workload included in the training set, of the KPI achieved by using *all* the available TM configuration $C_i \in C_1, \dots, C_K$. In other words, RD assumes a dense UM in the training set, except of course, for the workload being queried for which only a small subset of configurations is assumed to have been sampled. It starts by normalizing the dense UM with respect to a configuration C_i (line 2), and collects the maximum performance of each workload , building the vector M_w that contains the maximum values on a per workload basis (line 3). Rating distillation selects as "pivot" column C^* for the normalization of the whole UM, the one whose corresponding M_w has the smallest coefficient of variation.

Note that not only does this function reduce the numerical heterogeneity of ratings; it ensures the distance between two configurations is correctly encoded when the ratings are normalized (property *(i)*). Also, minimizing the dispersion of the maximum values allows to align the upper extreme of the rating distributions of each application (i.e., matrix row) to a common value: the tighter the distribution around a common value Mw, the closer it approximates an ideal "omniscient" normalization (property *(ii)*).

3.1.3 RecTM Workflow

After addressing how the recommender tackles heterogeneity, the next step is to understand how does RecTM optimize PolyTM (see Algorithm 2).

Algorithm 2 RecTM work-flow [6]

1: Off-line performance profiling of an initial training set of applications
2: Rating distillation and construction of the Utility Matrix.
3: Selection of CF algorithm and setting of its hyper-parameters.
4: Upon the arrival of a new workload:
5: Sample the workload on a small set of initial configurations.
6: Recommend the optimal configuration).

RecTM employs a black-box approach that relies on on-line and off-line training. Firstly it does off-line performance profiling of an initial training set, in which it explores a mix of workloads on the full spectrum of the available TM configurations. Next, it applies the rating distillation to obtain homogeneous ratings for each workload, resulting in the initial/training dense UM. Based on the training UM, the sub-module *Recommender*, selects and configures the CF algorithm (choosing between KNN and SVD) to use at run-time, cross-validation. Now that the off-line configurations ended, the system is ready to receive new workloads.

At the arrival of a new workload, the Controller drives the on-line profiling using Sequential Model-based Bayesian Optimization (SMBO). This technique samples the new workload in a small number of initial configurations, and tries to fit a probabilistic model. Then, it identifies the next point (TM configuration) to be sampled based on the expected gain computed on the basis of the CF-based recommendations (Expected Improvement [15]).

Based on the selected CF algorithm, it recommends the optimal configuration for the new workload. The *Monitor* is responsible for collecting the target KPI at the arrival of new workloads and feed it to the *Controller*, so that the later can realize the on-line profiling. The Monitor is also responsible for the detection of workload changes and for triggering the optimization process.

3.2. ProteusTM Limitations

As stated, Collaborative Filtering techniques in ProteusTM, bases its estimations assuming the existence of a fully populated *UM*. This is only feasible in the order of a few hundreds of training workloads and possible configurations. In fact, adding

a new tuning parameter in ProteusTM would lead to an exponential growth in the number of possible configurations (the so-called curse of dimensionality [3]). Hence, it would make the use of ProteusTM's RD technique impractical or even prohibitively onerous/time consuming.

3.2.1 Scalability of the normalization

The first extension we apply to ProteusTM is the redesign of the UM to consider sparsity. A tightly intertwined problem is to identify normalization techniques alternative to RD that do not assume the availability of a dense UM.

The *first part* of this thesis targets precisely this problem by proposing two normalization techniques: Box-Cox and Sparse Rating Distillation.

3.2.2 Lack of workload characterization info

A common technique adopted in the CF literature to cope with sparse UM matrix is to incorporate in the knowledge base not solely the explicit users ratings of the various items, but also a characterization of the user's profile.

This class of approaches motivates the research direction that this dissertation investigates: to what extent can the availability of information on the workload characteristics (e.g., transaction duration or abort rate) enhance the accuracy of RecTM, which, we recall, relies solely on KPI information. The key idea here is to extend the UM used in ProteusTM to incorporate workload information, i.e., extending the UM to include columns containing workload characteristics, information traceable at run-time.

In summary in this report we intend to answer the following research questions:

1. How can ProteusTM be extended to exploit a sparse training set/UM?

2. Would incorporating workload information improve ProteusTM accuracy?

4. ProteusTM Extension: Sparsity

This chapter will address the scalability problem of ProteusTM by investigating the problem of how to support sparse UM matrices in the ProteusTM framework.

As already mentioned, the base CF algorithms (SVD and KNN) employed by ProteusTM are already equipped to cope with sparse UM matrices. The key problem to address, though, is how to adapt the normalization procedure that generates the UM that is then fed to the CF algorithms, since RD assumes the use of a dense matrix.

In the following we propose and evaluate the use of two normalization schemes designed to cope with sparse matrices namely: Box-Cox (BC) and Sparse Rating Distilation (SRD). *Box-Cox* transformation, a well known data transformation technique [25] that has been already successfully used in other (non-TM related) optimization problems [29] and that aims to generate data that follows normal distributions. And the second SRD, namely a variant of the RD procedure, that uses a normalization technique similar in spirit to RD but adapted to cope with sparse matrices.

4.1. Box-Cox

Box-Cox is a parametric power data transformation, proposed in 1964, used in order to reduce anomalies such as non-normality in data [25].

The work in [29] used this transformation in order to stabilize data variance and make the data more normal distribution-like to fit their matrix factorization technique. Similarly in our own context, Box-Cox will be used to normalize the original UM.

Box-Cox is defined as follows, when the original data set $S \in R^+$:

$$boxcox(y) = \begin{cases} \frac{(y^\lambda - 1)}{\lambda}, & \text{if } \lambda \neq 0 \\ log(y), & \text{if } \lambda = 0 \end{cases} \quad (1)$$

where the parameter λ controls the extent of the transformation. The quality of the normalization is defined as the ability to map the data to a normal distribution. Unfortunately, the value of λ that gives the best normalization varies depending on the data set. Thus, a first challenge that we have tackled is to find a procedure to find the best λ. The procedure considered to tune parameter λ is based on the Kolmogorov-Smirnov test.

4.1.1 Kolmogorov-Smirnov Test

The Kolmogorov-Smirnov Test (KS-Test) is a non-parametric test that tries to determine if two data sets differ significantly from one another. We will use KS-Test as a "test of goodness of fit" which is concerned with the agreement between the distribution of a set of sample values and a theoretical distribution.

Suppose that a population is thought to have some specified cumulative frequency distribution function, say $F_0(x)$, i.e, for any specified value of x, the value of $F_0(x)$ is the fraction of observations in the population having measurements less than or equal to x. If $F_0(x)$ is the population cumulative distribution, and $SN(x)$ the observed cumulative step-function of a sample then the Kolmogorov-Smirnov statistic is [18]:

$$Dn = max|F_0(x) - SN(x)| \quad (2)$$

This statistic quantifies a distance between the empirical distribution function of the sample and the cumulative distribution function of the reference distribution. If the sample $SN(x)$, comes from a distribution close to $F_0(x)$ then the value of Dn will converge to 0.

We will use the Kolmogorov-Smirnov statistic in order to chose which is the best alpha for a training set in an automatic way.

4.1.2 Box-Cox and KS-Test

The algorithm that uses KS-Test to tune the parameter λ of Box-Cox will be presented in this section. The main objective of the algorithm is to find a value for λ that better normalizes a random training set, which is the one that minimizes the Kolmogorov-Smirnov statistic. We will use the one-sample KS-Test which will compare the empirical cumulative distribution of the matrix UM against a Normal Distribution with the same average and standard deviation.

Before explaining the algorithm it is relevant to mention that, the Box-Cox algorithm standard values for λ are normally in between the interval [-2,2] [13].

In order to use KS-Test together with Box-Cox this was the developed algorithm:

Algorithm 3 Box-Cox function in ProteusTM

1: **for** $\lambda \in [-2; 2]$; with step 0.1 **do**
2: Normalize Matrix KPI using box-cox with λ (equation 1).
3: Gather the new average (avg) and standard deviation (stdev).
4: Transform the resulting matrix into a vector v organized in an ascending order of its elements.
5: Apply KS-Test:
6: Build the reference NormalDistribution(avg, stdev), F_0.
7: Build the Empirical Cumulative Distribution (ECD) of v, $SN = ECD(v)$
8: Compute Dn for the current lambda(§ 4.1.1 equation 2)
9: **end for**
10: **return** the value λ^* that achieved minimum Dn

The first step to implement this algorithm is to apply the Box-Cox normalization (Eq. 1) on the UM, in an interval starting from -2 to 2, increasing by a fixed step (which we set to 0.1) per iteration (line 1-2). In order to use the KS-Test we need the reference Normal Distribution and the Empirical Cumulative Distribution (ECD) of our sample (UM). To build the reference Normal Distribution, we first calculate the average and the standard deviation of the elements present in the resulting matrix, and using the Apache Commons Library [1] we create the reference *NormalDistribution(average, standard deviation)*. Next, we transform the normalized UM into a vector, of size n, organized in ascending order and calculate its ECD.

With these two distributions we can calculate the Dn (line 8), repeating these steps until we reach the end of loop. Finally, the returned λ^* is the one that achieves the smallest Dn value (line 10) amongst the several iterations. To further enhance accuracy, once a λ^* has been found we repeat the procedure by zooming in in the neighbourhood of λ^*: $[\lambda^* - 0.1, \lambda^* + 0, 1]$. Repeating the search at a finer granularity (0.01).

4.2. Sparse Rating Distilation

RD assumes that rows in the training set can be normalized using the value of any configuration. With a sparse training set, however, it is possible that some workloads cannot be normalized by a given configuration, because that configuration has not been sampled. SRD modifies RD to cope with sparse training data.

SRD is our modified version of the RD that works very intuitively. In short, for each configuration C_i, SRD builds a sub-matrix of the UM of size $m \times C_K$, where only the rows that store some KPI value for column C_i are present, i.e, rows missing the rating for column C_i are removed. Then, RD is performed on each sub-matrix against C_i, and the C_i delivering the lowest coefficient of variation is chosen.

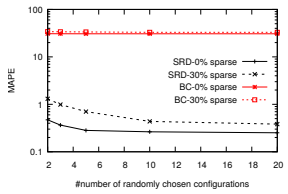
In case of high sparsity, however, it can happen that a sub-matrix is extremely small, because a given configuration has been sampled by a very limited number of training workloads. When this happens, the statistical meaning of any results obtained with such a matrix is very limited, and can even be misleading, leading to bad normalizations.

To cope with this issue we augment SRD with a simple threshold-based predicate to evaluate whether a sub-matrix has enough data to be considered statistically meaningful.

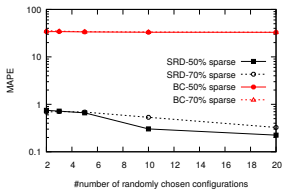
Let p the percentage of sparsity in the UM, m the workloads selected to be part of the sub-matrix and w the number of workloads in the original matrix. The threshold is the following: $|m| >= |w| * (1.0 - p)$, i.e., the number of rows of the sub-matrix cannot be less than $(1.0 - p)$ of the original matrix.

The new algorithm chooses the first configuration C^* that minimizes the index of dispersion M_w and respects the threshold.In case the sparsity is so high that no valid sub-matrix can be found we use the return statement of the original algorithm.

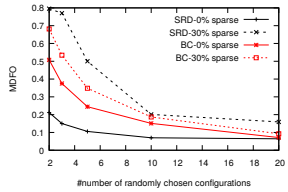
[1] http://commons.apache.org/



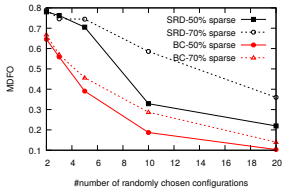
(a) MAPE for 0% and 30% sparsity



(b) MAPE for 50% and 70% sparsity



(c) MDFO for 0% and 30% sparsity



(d) MDFO for 50% and 70% sparsity

Figure 4: KNN - MAPE and MDFO for different sparsity level of a 30% training set.

4.3. Evaluation

In this sections we compare both normalization techniques for different sparsity levels, utilizing the same data set and methodology as in the previous study.

Figure 4 (a) and 4 (b) shows that BC performs poorly in terms of prediction accuracy, which does not improve when increasing the number of sampled configurations per workloads. In contrast, SRD attains a better MAPE. The quality of predictions in SRD increases when adding additional sampled workloads and naturally decreases as the sparsity increases. However even at 70% sparsity, the MAPE is in average only 40% worse than the MAPE achieved with a dense matrix.

Figure 4 (c) and 4 (d) report the MDFO for different sparsity levels. The results for 0% sparsity SRD always beats BC, as we would expect looking at the MAPE. The results changes when sparsity is introduced. Although, the MAPE results for BC are really poor, it predicts with good accuracy the optimal configurations (Figure 4(c) and (d)). BC with little knowledge about the workloads (2,3 initial configurations) obtains an MDFO around the 50%-70%, while SRD obtains values firmly around the 80%. Regarding the best results, when we sample 20 initial configurations BC reaches an MDFO around 10% while SRD considering only sparsity > 0 oscillates between 20%-48%.

Although MAPE for BC is much worse than SRD, BC is able to allow Proteus to identify a better configuration. We argue this is because BC can correctly rank the quality of the configurations, so as to be able to pick the best. The absolute prediction error is not very important, as long as the ranking of the configurations correspond to the ground truth. Instead, for example, the MAPE can be low because the predictor is able to predict very well the performance for very bad configurations.

This study indicates that, with sparsity, BC might be a better choice for TM optimization. However, given its better MAPE, SRD would be better for QoS-aware selftuning i.e., to adapt the TM to match a specific performance requirement, rather than just deployment the best configuration possible.

When looking at the MDFO metric, RD generally outperforms BC in absence of sparsity. When sparsity is introduced, though, BC achieves significantly higher accuracy levels than SRD.

5. ProteusTM Extension: Workload Characteristics

A vast body of literature is devoted to enhance the accuracy of RS by incorporating additional information regarding the profile of the user (e.g., age and sex), of the items (e.g., genre of a movie) and/or context in which users interact (e.g., click or purchase history) [26, 23, 22].

In this section we present a study on the impact of adding additional information to the original UM of ProteusTM. More precisely, this study assess the idea of including in the UM, besides the KPIs achieved by the various workloads across the various configurations (which corresponds to the user's ratings for the various items) also different workload characteristics, including average aborts, maximum retries, writes duration.

The first step was to extend ProteusTM, in order to support a new type of UM the Extended UM (EUM). EUM is obtained by merging, on a per-workload basis, ProteusTM's original UM with another data-set containing workload characteristics.

A key conceptual problem at the basis of this approach is that it leads to blend in the same row of the UM different types of information, which can be expressed using completely heterogeneous scales. In fact, while the values stored in the original UM refer to the same KPI/metric (although expressed workload-dependant scales), the EUM contains numerical values associated with very diverse domains (e.g., abort rate, transaction duration and throughput). Hence, a relevant problem to address in order to jointly use KPIs and workload characteristics is how to ensure that the information encoded in the EUM can be meaningfully interpreted by a RS.

We will consider two possible approaches to tackling this problem:

(1) using different normalization schemes feed the resulting matrix to the CF-based algorithms used by ProteusTM, e.g. SVD or KNN.

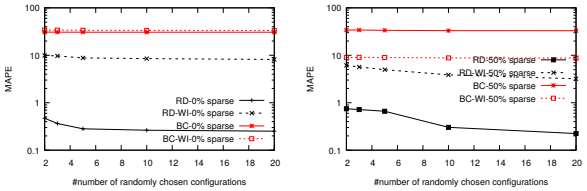
(2) using LibFM [23], a RS software tool in-

tegrated with ProteusTM, based on Factorization Machines (FMs) designed to support features comprising values belonging to different domains and expressed using heterogeneous scales.

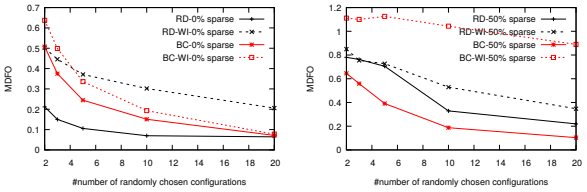
5.1. Evaluation

The objective of this section is to answer this question: how is ProteusTM effectiveness impacted by the addition of workload info, with sparse training data?

This study was conducted using the same training sets as in the previous sections, but do to space constraint we only present the results for 0% and 50% sparsity. The workload features have been selected taking into account previous studies [6, 28] and further refined through a feature selection phase, using Pearson product-moment correlation coefficient (PCC) and Information Gain (IG). IG was utilized in order to output a feature ranking, in which it suggested the most informative features for our training set. This filtering outputted 11 workload characteristics, which we use to obtain the results that are now presented.



(a) MAPE - 0% sparsity (b) MAPE - 50% sparsity

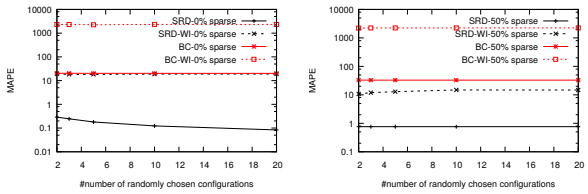


(c) MDFO - 0% sparsity (d) MDFO - 50% sparsity

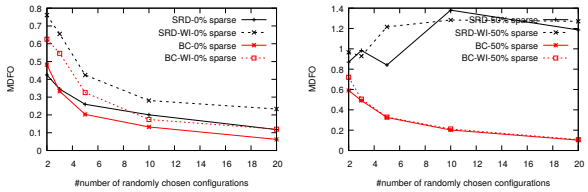
Figure 5: KNN - Impact of using EUM for sparsity levels 0% and 50%

This evaluation starts by studying the impact on one of the CF techniques, namely KNN (Figure 5). Next, we present the same study with LibFM (Figure 6). We note SRD-WI, BC-WI as the normalization variant that also take into account workload info.

The results lead to the following conclusion: at least for the considered information fusion approaches and datasets, the use of additional workload information degrades, rather than enhancing, the accuracy of typical RS algorithms, in particular KNN and LibFM.



(a) MAPE - 0% sparsity (b) MAPE - 50% sparsity



(c) MDFO - 0% sparsity (d) MDFO - 50% sparsity

Figure 6: LibFM - Baseline and workload information for sparsity levels 0% and 50%

6. Conclusions

This dissertation builds on a recent self-tuning system for TM, called ProteusTM, which has a unique feature in the literature: it is the only self-tuning solution for TM systems that supports dynamic optimization across a *multi-dimensional* configuration space.

In particular, this dissertation investigates two key research questions : i) how to extend ProteusTM to support sparse training sets, and ii) to what extent can the inclusion of workload characteristics (e.g., abort rate) enhance the accuracy achieved by ProteusTM's. We answered the first question by proposing and evaluating the use of two alternative normalization techniques, based on the Box-Cox data transformation and on a novel technique, which we called Sparse Rating Distillation (SRD).

The results highlight that BC is the best technique for ProteusTM, when sparse training sets are used. BC even in very sparse scenarios can achieve results close to the optimum (around 10% from optimal) with both learners. In comparison, with sparsity SRD does not perform as well as BC due to the fact that it forces a drop in the percentage of the training set equal, on average, to the percent of sparsity of the whole data set.

As for the second question, in contrary to our initial expectations, our results suggest that the inclusion of workload information clouds, in a remarkably consistent way, the predictor's accuracy for all the considered data fusion, normalization and learning techniques.

References

[1] A.-R. Adl-Tabatabai, C. Kozyrakis, and B. Saha. Unlocking concurrency. *Queue*, 4(10):24–33, 2006.

[2] C. M. Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006.

[3] K. L. Clarkson. An algorithm for approximate closest-point queries. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 160–164. ACM, 1994.

[4] L. Dalessandro, F. Carouge, S. White, Y. Lev, M. Moir, M. L. Scott, and M. F. Spear. Hybrid norec: A case study in the effectiveness of best effort hardware transactional memory. *ACM SIGARCH Computer Architecture News*, 39(1):39–52, 2011.

[5] L. Dalessandro, M. F. Spear, and M. L. Scott. Norec: streamlining stm by abolishing ownership records. In *ACM Sigplan Notices*, volume 45, pages 67–78. ACM, 2010.

[6] D. Didona, N. Diegues, R. Guerraoui, A.-M. Kermarrec, R. Neves, and P. Romano. Proteustm: Abstraction meets performance in transactional memory. In *Twenty First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016.

[7] D. Didona, P. Felber, D. Harmanci, P. Romano, and J. Schenker. Identifying the optimal level of parallelism in transactional memory applications. *Computing*, 97(9):939–959, 2015.

[8] N. Diegues and P. Romano. Self-tuning intel transactional synchronization extensions. In *11th International Conference on Autonomic Computing (ICAC 14)*, pages 209–219, 2014.

[9] N. Diegues, P. Romano, and L. Rodrigues. Virtues and limitations of commodity hardware transactional memory. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 3–14. ACM, 2014.

[10] M. D. Ekstrand, J. T. Riedl, J. A. Konstan, et al. Collaborative filtering recommender systems. *Foundations and Trends® in Human–Computer Interaction*, 4(2):81–173, 2011.

[11] P. Felber, C. Fetzer, and T. Riegel. Dynamic performance tuning of word-based software transactional memory. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 237–246. ACM, 2008.

[12] R. Guerraoui, M. Kapalka, and J. Vitek. Stmbench7: a benchmark for software transactional memory. Technical report, 2006.

[13] J. Hintze. Ncss statistical software. *NCSS, Kaysville, UT*, 1998.

[14] A. G. Holla and M. Herlihy. Lock elision for memcached: Power and performance analysis on an embedded platform. *Computer Science Department, Brown University*, pages 1–9, 2013.

[15] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

[16] G. Kestor, R. Gioiosa, T. Harris, O. S. Unsal, A. Cristal, I. Hur, and M. Valero. Stm2: A parallel stm for high performance simultaneous multithreading systems. In *Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on*, pages 221–231. IEEE, 2011.

[17] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8), 2009.

[18] F. J. Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American statistical Association*, 46(253):68–78, 1951.

[19] C. C. Minh, J. Chung, C. Kozyrakis, and K. Olukotun. Stamp: Stanford transactional applications for multi-processing. In *Workload Characterization, 2008. IISWC 2008. IEEE International Symposium on*, pages 35–46. IEEE, 2008.

[20] T. M. Mitchell. Machine learning. *Machine Learning*, 1997.

[21] T. Nakaike, R. Odaira, M. Gaudet, M. M. Michael, and H. Tomari. Quantitative comparison of hardware transactional memory for blue gene/q, zenterprise ec12, intel core, and power8. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 144–157. ACM, 2015.

[22] S. Rendle. Factorization machines. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 995–1000. IEEE, 2010.

[23] S. Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[24] D. Rughetti, P. Di Sanzo, B. Ciciani, and F. Quaglia. Machine learning-based self-adjusting concurrency in software transactional memory systems. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2012 IEEE 20th International Symposium on*, pages 278–285. IEEE, 2012.

[25] R. Sakia. The box-cox transformation technique: a review. *The statistician*, pages 169–178, 1992.

[26] Y. Shi, M. Larson, and A. Hanjalic. Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys (CSUR)*, 47(1):3, 2014.

[27] X. Su and T. M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009:4, 2009.

[28] Q. Wang, S. Kulkarni, J. Cavazos, and M. Spear. A transactional memory with automatic performance tuning. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4):54, 2012.

[29] J. Zhu, P. He, Z. Zheng, and M. R. Lyu. Towards online, accurate, and scalable qos prediction for runtime service adaptation. In *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*, pages 318–327. IEEE, 2014.