Self-tuning in Distributed Transactional Memory

Maria Couceiro, Diego Didona, Luís Rodrigues, and Paolo Romano

INESC-ID, Instituto Superior Técnico, Universidade de Lisboa,
Lisbon, Portugal

Abstract. Many different mechanisms have been developed to implement Distributed Transactional Memory (DTM). Unfortunately, there is no "one-size-fits-all" design that offers the desirable performance across all possible workloads and scales. In fact, the performance of these mechanisms is affected by a number of intertwined factors that make it hard, or even impossible, to statically configure a DTM platform for optimal performance. These observations have motivated the emergence of self-tuning schemes for automatically adapting the algorithms and parameters used by the main building blocks of DTM systems. This chapter surveys existing research in the area of autonomic DTM design, with a focus on the approaches aimed at answering the following two fundamental questions: how many resources (number of nodes, etc.) should a DTM platform be provisioned with, and which protocols should be used to ensure data consistency.

1 Introduction

After more than a decade of research, implementations of the Transactional Memory (TM) abstraction have matured and are now ripe to enter the realm of mainstream commodity computing. Over the last couple of years, TM support has been integrated in the most popular open-source compiler, GCC, and also in the CPUs produced by industry-leading manufacturers such as Intel [1] and IBM [2]. Distributed Transactional Memory (DTM) [3,4,5] represents a natural evolution of this technology, in which transactions are no longer confined within the boundaries of a single multi-core machine but, instead, may be used as a synchronization mechanism to coordinate concurrent executions taking place across a set of distributed machines. Just like TM have drawn their fundamental motivation in the advent of multi-core computing, the need for identifying simple, yet powerful and general programming models for the cloud is probably one of the key factors that have garnered growing research interest in the area of DTM over the last years [6]. Another major driver underlying existing research efforts in the area of DTM is fault-tolerance: as TM-based applications are expected to turn mainstream in the short term, it becomes imperative to devise efficient mechanisms capable of replicating the state of a TM system across a set of distributed nodes in order to ensure their consistency and high-availability despite the failures of individual nodes [7,8].

R. Guerraoui and P. Romano (Eds.): Transactional Memory, LNCS 8913, pp. 418–448, 2015.

From the existing literature in the area of DTM, it can be observed that the design space of DTM platforms is very large and encompasses many complex issues, such as data placement and caching policies, replication protocols, concurrency control mechanisms, and group communication support, just to name a few. The performance of these fundamental building blocks of a DTM is affected by multiple intertwined factors. This has motivated the development of a wide range of alternative implementations, each exploring a different trade-off in the design space and optimized for different workload types, platform's scales, and deployment scenarios. As a result, the body of literature on DTM encompasses solutions tailored for read-intensive [7] vs conflict-prone [9,10] workloads, replication mechanisms optimized for small clusters [11], large scale data centers [12,13], as well as approaches specifically targeting geographically distributed DTM platforms [3].

One of the key conclusions that can be easily drawn by analyzing the results above is that there is no "one-size-fits-all" solution that can provide optimal performance across all possible workloads and scales of the platform. This represents a major obstacle for the adoption of DTM systems in the cloud, which bases its success precisely in its ability to adapt the type and amount of provisioned resources in an elastic fashion depending on the current applications' needs. Besides, a DTM encompasses an ecosystem of complex subcomponents whose performances are governed by a plethora of parameters: manually identifying the optimal tuning of these parameters can be a daunting task even when applications are faced with static workloads and fixed deployments. Guaranteeing optimal efficiency in presence of a time varying operational envelope, as typically occurs in cloud computing environments, requires to adjust these parameters in a dynamic fashion — a task that is arguably extremely onerous, if not impossible, without the aid of dedicated self-tuning mechanisms.

This is precisely the focus of this chapter, in which we dissect the problem of architecting self-tuning mechanisms for DTM platforms, with a special emphasis on solutions that tackle the following two fundamental issues:

- *elastic scaling:* DTM systems can be deployed over platforms of different scales, encompassing machines with different computational capacities interconnected via communication networks exhibiting diverse performances. Hence, a fundamental question that needs to be addressed when architecting a DTM-based application is how many and what types of resources (number of nodes, their configuration, etc.) should be employed (e.g., acquired from an underlying IaaS (Infrastructure as a Service) cloud provider) in order to ensure predetermined performance and reliability levels. In cloud computing environments, where resources can be dispensed elastically, this is not a one-off problem, but rather a real-time optimization problem. Its optimal solution requires not only to estimate the performance of applications when deployed over infrastructures of different scale and types, but also to encompass economical aspects (e.g., by comparing the cost of a DTM deployment over a large number of relatively slow nodes against a deployment on a smaller number of more powerful machines) as well as issues related to the on-line reconfiguration of the platform (namely, how to rearrange data after scaling);

- *adapting the data consistency protocol:* the literature on data consistency protocols for distributed and replicated transactional systems is a quite prolific one. Existing approaches explore a number of different design choices, concerning aspects such as whether to execute transactions on all nodes (as in active replication [14]) or executing in just one replica and only propagating the transaction's updates (a.k.a. deferred update schemes [15]), how to implement transaction validation [16], and whether to use distributed locking [17] vs total order communication protocols [18] to serialize transactions. This has motivated research aimed at supporting the automatic switching between multiple data consistency protocols, and, in some cases even the simultaneous coexistence of different protocols. The key challenges addressed in these works are related to how to preserve consistency despite the (possibly concurrent) employment of alternative consistency protocols, as well as to the identification of the best strategy to adopt given the current workload and system's characteristics.

The remainder of this chapter is structured as follows. We first provide, in Section 2, an overview of the main building blocks encompassing typical DTM architectures, and illustrate some of the key choices at the basis of their design. Next, in Section 3, we identify the DTM components that would benefit the most from the employment of adaptive, self-tuning designs. In Section 4, we provide background on the main methodologies employed in the literature to decide when to trigger an adaptation and to predict which among the available strategies to adopt. In Section 5 we focus on elastic scaling, and in Section 6 we discuss adaptation of the consistency protocols. Finally, Section 7 concludes the paper.

2 Background on DTM

This section is devoted to overview on the key mechanisms that are encompassed by typical DTM architectures. It should be noted that the discussion that follows does not aim at providing a thorough and exhaustive survey of existing DTM designs, but rather to facilitate the description of the self-tuning DTM systems described in the remainder of this chapter.

The diagram in Figure 1 depicts the high level architecture of a typical DTM platform, illustrating the key building blocks that compose the software stack of this type of system.

DTM API. At their top most layer, existing DTM platforms expose APIs analogous to those provided by non-distributed TMs that allow to define a set of accesses to in-memory data to be performed within an atomic transaction. The actual API exposed by a DTM is ultimately influenced by the data model that it adopts; the range of data models explored in the DTM literature includes, besides the object-based [7] and word-based [5] ones (typically employed

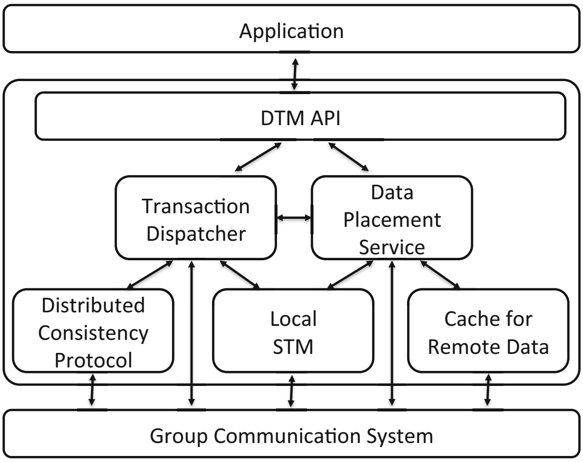

Fig. 1. High level architecture of typical DTM platforms (single node)

in non-distributed TMs), also popular alternatives in the NoSQL domain, like the key-value [13,19] model. Certain DTM platforms [20,21] that support partial replication schemes (i.e., which do not replicate data at *every* replica of the system) provide also dedicated API support to influence the policies employed to determine the placement of data (and its replicas) across the nodes of the system, with the goal of enhancing the data locality achieved by DTM applications. These include programmatic mechanisms to ensure the co-location of data items [21] or to provide the data placement service with semantic information (like the data item's type and the relations in which it is involved) concerning the data access patterns generated by the nodes of the platform [20].

Data Placement Service. The data placement service, as the name suggests, is responsible for locating the nodes that maintain (replicas of) the data items accessed during the transaction execution. This module is required exclusively in case the DTM platform adopts a partial replication scheme (as in fully replicated systems each node maintain a replica of every data item), although certain DTM platforms may rely on analogous abstractions to establish ownership privileges of nodes on data items [21]. The actual implementation of this service is strongly affected by the transaction execution model embraced by the DTM, which can be either *control-flow* or *data-flow*. In *control-flow* systems data items are statically assigned (unless the platform is subject to elastic scaling) to the nodes of the platform, which retrieve non-local data items via RPC. In *data-flow* systems, conversely, transactions are immobile and objects are dynamically migrated to invoking transactional nodes. As in the control-flow model the placement of data is static, several control-flow DTM systems [21,22,12] adopt simple policies based on consistent hashing [23]. This technique, which essentially maps data items to nodes of the platform randomly via the use of a hash function, has the desirable properties of executing data items look ups locally (i.e., the nodes that replicate a given data item can be identified by computing the hash of its identifier) and achieving a good balance in the data distribution. Data-flow DTMs, on the other hand, rely on ad-hoc (distributed) directory or cache coherence protocols, such as the Arrow [24] or the Ballistic [25] protocols. These protocols require that, in order for a node to access a data item, it must first acquire its ownership (which implies locating the current data item owner). As a result, data-flow models can introduce additional network hops along the critical path of execution of transactions with respect to control-flow solutions (that do not allow migration of data). On the pro-side, by dynamically moving the ownership of items to the nodes that access them, data-flow systems can spontaneously lead to data placement strategies achieving better locality than static policies, like consistent hashing, supported exclusively by control-flow systems. A detailed discussion on control-flow and data-flow models, as well as on systems adopting these models, can be found in Chapter 16.

Transaction Dispatcher. The transaction dispatcher is a component present in several DTM platforms [10,5,26], and is in charge of determining whether the execution of a transaction should take place on the node that generated it, on a different one, or even by all nodes in the platform. This decision can be driven by different rationales, such as reducing data contention [26] or enhancing data locality [10,5,21]. In order to support the migration and execution of entire transactions at remote nodes, the transaction dispatching mechanism typically requires ad-hoc support at the DTM API layer in order to ensure proper encapsulation of the transaction logic, i.e., a function/procedure encoded in a programming language, and of its input parameters (using classic RPI mechanisms).

Local STM. As for the local data stores, existing DTM platforms typically leverage on state of the art local STMs, which implement efficient concurrency control algorithms optimized for modern multi-core architectures [7,11,9,27].

Cache for Remote Data. Some partially replicated DTM platforms [28,21] cache frequently accessed remote data items, and update them using lazy/asynchronous invalidation strategies. Clearly, it must be possible to manipulate also cached data without breaking consistency: therefore they are maintained in memory and their manipulation is subdued to some form of concurrency control. However, cached data need typically to be associated with different meta-data and managed with different rules than the data stored in the local STM (whose ownership can be established via the data placement service). As a consequence, cached data are normally maintained in separate in-memory structures.

Distributed Consistency Protocol. Clearly, the data accesses performed by local transactions need to be synchronized with those issued by transactions

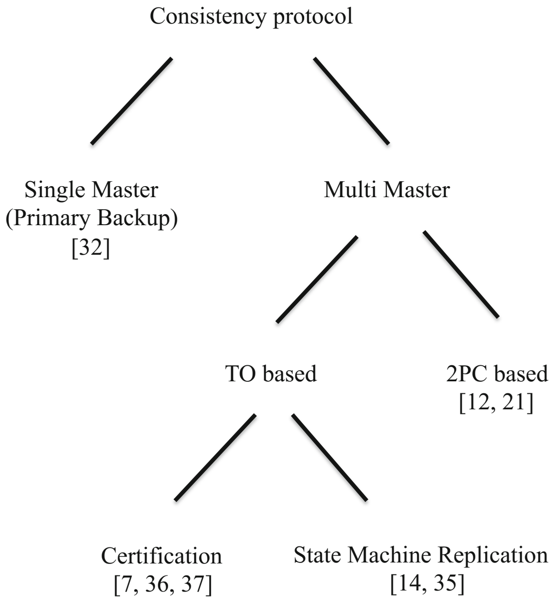
Consistency protocol
Single Master (Primary Backup) [32]
Multi Master
TO based
2PC based [12, 21]
Certification [7, 36, 37]
State Machine Replication [14, 35]

Fig. 2. Taxonomy for consistency protocols in transactional systems

executing at different nodes. The responsibility of this task is delegated to a distributed consistency protocol, which is ultimately responsible for enforcing the consistency guarantees ensured by the DTM platform. The literature on DTM (and more in general on distributed transactional platforms, e.g., distributed DBMS) has explored a number of alternative consistency levels, like 1-copy serializability [13], virtual world consistency [9], extended update serializability [12] and parallel SI [29]. Clearly, the choice of the consistency criterion has a strong impact on the design of the underlying distributed consistency protocol. Another factor that has a key impact on the distributed consistency protocol is whether the system employs full or partial replication. In fully replicated DTM platforms, in fact, once the transaction serialization order is established (typically by means of a consensus or atomic broadcast service [7]), the nodes can determine the outcome of committing transactions locally (by validating their read-set with respect to the most recent committed version). Conversely, in partially replicated DTM systems, some sort of 2PC-like agreement is unavoidable, as the snapshot accessed by a committing transaction needs to be validated, in general, by multiple nodes, which must certify the freshness of the transaction's snapshot with respect to the locally stored authoritative copies of data. Over the last decades, a vast literature on distributed consistency protocols for transactional systems has emerged [15,30,31]. A possible taxonomy of existing solutions is reported in Figure 2.

Single-master. In single master schemes, also known as primary backup, write transactions are executed exclusively at a single node (also called master or primary), whereas the remaining replicas can only run read-only transactions [32]. Upon failure of the master, a backup replica is elected to become the new master.

Note that, as the write transactions can be serialized locally by the master using its local concurrency control algorithm, this approach can rely on a simpler replica synchronization scheme with respect to multi-master solutions (as we will see shortly). On the down side, the throughput of write transactions does not clearly scale up with the number of nodes in the system, which makes the master prone to become the system bottleneck.

Multi-master. Multi-master schemes, on the other hand, are typically more scalable as transactions can be processed on all nodes. There are two types of synchronizing the accesses to data: eager and lazy. The first relies on a remote synchronization phase upon each (read/write) access, which normally results in very poor performance results [33].Conversely, the lazy approach defers replica synchronization till the commit time, which is when the transaction is finally validated. Lazy multi-master schemes can be classified based on whether they rely on Atomic Commit Protocols (such as Two-Phase Commit) or Total Order (TO) [34] broadcast/multicast schemes to determine the global serialization order of transactions.

Two-Phase Commit. In solutions based on Two-Phase Commit (2PC), transactions attempt to atomically acquire locks at all nodes that maintain data accessed by the transaction. Even though these schemes normally incur in minor communication overheads with respect to those relying on TO, these solutions are well known to suffer of scalability problems due to the rapid growth of the distributed deadlock rate as the number of replicas in the system grows [17].

Total Order based schemes. Conversely, TO-based replication is a family of (distributed) deadlock-free algorithms that serializes transactions according to the total order established by a TO service [34]. These solutions can be distinguished into two further classes: state machine replication and certification.

State Machine Replication. In the state machine replication [14,35], all replicas[1] execute the same set of transactions in the same order. The transactions are shipped to all replicas using total order broadcast and, consequently, all replicas receive transactions in the same order and execute them in that order. However, both transactions and validation scheme must be fully deterministic so that all replicas begin and finish transactions in the same state.

Certification. Unlike State Machine Replication, certification based techniques undertake a *speculative* approach, which can achieve higher scalability, in low conflict workloads, by fully executing the transaction only at one node. This means that different transactions may be executed on different replicas concurrently. If the transaction aborts during its execution, no further coordination is

[1] This technique has been proposed for fully replicated systems.

required. However, if the transaction is ready to commit, a transaction validation phase is triggered in order to certify that it has not accessed stale items. The information exchanged to certify transactions varies depending on the considered certification protocol (e.g., non-voting [36], voting [37] or bloom-filter based [7]), but the certification request is disseminated by means of a TO broadcast service that targets all the nodes that maintain replicas of the data items accessed by the transaction. In case of partial replication, as already mentioned, this certification phase may have to involve a voting phase to gather positive acknowledgements from at least one replica of each data item accessed within the transaction; in this case the message pattern of the distributed consistency protocols coincides with the one of the 2PC scheme, in which the prepare messages are disseminated using a TO service.

3 What Should Be Self-tuned in a DTM?

As it clearly emerges from the discussion in the previous section, the design and configuration space of DTM is quite vast, and there are several components in the DTM stack whose setting and parametrization has a strong impact on DTM performance. Indeed, performance of a DTM application are driven by complex non-linear dynamics stemming from the intertwined effects of workload's resource utilization (e.g., in terms of CPU and network bandwidth), data access pattern (e.g., data contention and locality), inter-nodes communication (e.g., for remote read operations) and distributed synchronization (e.g., for committing transactions).

Typical Key Performance Indicators (KPIs) of a DTM are general purpose metrics like transactions response time and achievable throughput. DTM-specific KPIs include also metrics like transactions abort probability, execution time of the distributed commit phase, number of remote accesses during the execution phase, and number of nodes involved in the transaction processing. While Quality of Service specifications are typically expressed in terms of throughput and response time, DTM-specific KPIs are fundamental metrics in many DTM self-tuning schemes, as they allow for pinpointing bottlenecks and for identifying sub-optimal configurations. For example, a high abort rate may imply an excessive concurrency level in the platform and may lead to the decrease of the number of concurrently active transactions in the platform.

Recent research [26,38,39,40] has shown that transactional workloads are very heterogeneous and affected by so many variables that no-one-size-fits-all solution exists for the DTM configuration that guarantees optimal performance across all possible applications' workloads. To address this issue, a number of alternative solutions have been proposed to tackle the problem of self-tuning DTMs. Such solutions draw from different fields of performance modeling and forecasting and aim to optimize several major building blocks/configuration parameters of DTMs, focusing in particular on the following five aspects: elastic scaling, choice of the consistency protocol, data placement and replication degree, communication layer and local TM implementation.

In the following, we analyze the main trade-offs that emerge in the self-tuning of these DTM building blocks. In Section 5 and Section 6 we will return to investigate in greater detail the problems of automating the elastic scaling process and the choice of consistency protocol, by surveying existing research in these areas.

Scale. The scale of a DTM consists in the number of nodes composing the platform and, possibly, the maximum number of active threads allowed on each node, namely, the multiprogramming level (MPL). Accordingly, the elastic scaling, i.e., dynamic resizing, of a DTM can take place horizontally, by altering number of nodes in the platform, or vertically, by adapting the MPL.

Different scales in the DTM not only result in a different physical resources utilization, but also into different data access patterns. In fact, increasing the number of active transactions in the system, either by scaling horizontally or vertically the platform, other than requiring more processing power, also results into a higher concurrency in accessing and modifying shared data, with a possible commensurate increase of conflicts and, hence, abort rate. This poses a major challenge when devising elastic scaling schemes for DTMs as the bottleneck of a DTM application may lie in data contention. Hence, scalability trends of DTM applications are far from being easily predictable, as increasing the processing power, i.e., number of nodes, or processing units, i.e., number of threads, does not always entail better performance.

Scaling out a DTM poses additional challenges than altering its MPL level: changing the number of nodes composing a DTM, in fact, results not only into an increased processing power, but also into a modification of the placement of data, which can get redistributed across the nodes of the platform (as it is case, for instance, when using consistent hashing-based placement policies). Such modification can imply a shift in data locality, and affect the probability that a transaction accesses data maintained by its originating node. For write transactions this results also in a change in the number of nodes to be contacted at commit time to propagate updates and, hence, in the duration of the corresponding phase.

The aforementioned DTM dynamics are not encompassed by the vast majority of available state-of-the-art solutions for automatic resource provisioning, as they mainly target stateless applications or neglect the impact of elastic scaling on data distribution and contention [41,42,43,44,45,46]. Devising an optimal autonomic elastic scaling schemes for DTM is, thus, a very challenging task, which needs to be tackled by means of *ad hoc* solutions.

Distributed Consistency Protocol. Like for the scale, the choice of the distributed consistency protocol has a huge impact on both logical and physical resource utilization. Single master approaches deal with the concurrency control of update transactions on the master node: on one side this tends to mitigate data contention, as conflicts can be resolved more efficiently, i.e., in a fully local fashion and without the need to run a distributed consensus algorithm to determine the outcome of a transaction; on the other hand, the master node may become a bottleneck in case the arrival rate of update transactions exceeds its processing capacity.

Multi-master schemes, instead, allow for a better load balancing among nodes even in write dominated workloads (by distributing update transactions across all the nodes of the DTM platform), but generally require onerous inter-node synchronization mechanisms for detecting and resolving conflicts among transactions. As mentioned in Section 2, consistency protocols based on 2PC require only two round-trip between a transaction's initiator and other involved nodes to agree on the outcome of the transaction, but are liable to distributed deadlocks; TO-based protocols, conversely, achieve deadlock freedom, but the latency induced by the TO primitive may lead to higher synchronization costs at commit time [39].

Data Placement and Replication Degree. Data locality plays a role of paramount importance in DTMs, as it determines the frequency of access to remote data present in the critical path of execution of transactions [20]. The tuning of the data placement and of the replication degree is aimed at enhancing the quality of the data layout, so as to increase data locality and reduce the execution time of transactions.

Two fundamental challenges that need to be tackled for implementing effective self-tuning data placement schemes are i) how to identify the optimal data layout (i.e., the data layout that maximizes the performance of the platform), and ii) how to keep track of the new mapping between data item replicas and nodes in the DTM platform. The former is in fact a distributed optimization problem, which has been addressed both in its on-line [20,47] and off-line [48,49] formulation, considering different objective functions and constraints (e.g., maximizing locality [20,48] vs balancing load [47]) and both centralized [48] and decentralized [20] solutions. As for the tracking of the mapping between data items and nodes of the DTM platform, there are two main trade-offs that need to be taken into account. Approaches relying on external (and properly dimensioned) directory services [48,47] can typically support fine-grained mapping strategies also for large data sets, but impose non-negligible additional latency in the transaction's critical path. Approaches that explicitly store the mapping of the entire data set at each node either rely on random hash functions [21] or on coarse grained mapping strategies — as the overhead for storing and keeping synchronized a fine-grained mapping would be unbearable with large data sets. This has motivated the usage of probabilistic techniques [20,49] that sacrifice accuracy of data items lookups in order to reduce the memory footprint of the meta-data used to encode the data-to-nodes mapping.

The tuning of the replication degree in a DTM [50,38] is another closely related problem, which encompasses a subtle trade-off between the probability of accessing locally stored data and the cost of the synchronization phase necessary to validate committing transactions. On one hand, in fact, increasing the replication degree generally results into a higher probability that a transaction accesses a data item that is maintained by the local node; on the other hand, for update transactions, it also typically leads to an increase in the number of nodes to be contacted at commit time for validating the transaction and propagating its updates [38].

Group Communication System. Inter-nodes communication represents a major source of overhead in DTM, as it can introduce relatively large latencies in the critical path of execution of transactions, both for the retrieval of remote data items and to support the distributed commit phase [4,51]. Other than increasing transactions' completion time (and hence reducing the achievable throughput), these latencies can have a great impact also on the conflict rate of transactions: in fact, the longer a transaction takes to execute, the higher is the chance that another transaction will try to concurrently access and/or modify a common datum.

A typical trade-off that arises in the design of coordination services, like consensus or total order multicast primitives, is that configurations/protocols that exhibit minimum latencies at low message arrival rate tend also to support relatively low throughputs. Conversely, protocols/configurations optimized for supporting high throughputs normally introduce much higher latencies when operating at low throughput levels. These trade-offs have motivated the development of self-tuning mechanisms supporting both the dynamic switching between alternative implementations of communication primitives (e.g., variants of TO) [52,53], as well as automatic configuration of internal parameters of these protocols (e.g., message batching) [54,55].

Local TM. As discussed in Section 4.2, the typical architecture stack of DTM systems includes a non-distributed (S)TM, which is used to regulate concurrent access to locally stored data. The problem of self-tuning TM has also been largely explored in literature, as TM and DTM, unsurprisingly, exhibit similar trade-offs, e.g., the workload characteristics can strongly affect the performance of the concurrency control algorithm, as well as the optimal MPL. Examples of self-tuning solutions that dynamically adjust these TM mechanisms/parameters can be found in [56,57,58,59].

Another TM parameter that has been object of self-tuning techniques is the lock granularity [60]. Lock granularity expresses what is the atomic portion of the data set (or of the memory space, for centralized TMs) that the concurrency control scheme deals with. The finer is the granularity, the higher is the concurrency that the concurrency control scheme allows for, but also the overhead incurred to maintain and manage meta-data. For example, in a per-item locking scheme, every data item is guarded by a lock and conflicts can be detected at the granularity of the single item. A coarser scheme, instead, reduces the number of employed locks at the cost of inducing false conflicts, i.e., conflicts among transactions that access different data items, which, nonetheless, insist on the same lock.

Finally, self-tuning techniques have been proposed to optimize the thread mapping strategy [61] and efficiently exploit the memory hierarchy of modern multiprocessors. In these architectures, just like we just described for the distributed case, data locality plays a fundamental role in determining the performance of an application. Thread mapping consists in placing threads on cores so as to amortize memory access latency and/or to reduce memory contention, i.e., it tries to allocate a thread that frequently accesses a given memory region on the core that incurs the minimal latency when accessing that portion of the memory space.

4 When and Which Adaptation to Trigger?

In this section, we provide background on the main methodologies that are commonly employed in the literature of self-tuning systems to tackle two key issues: when to trigger an adaptation, and how to predict which among the available reconfigurations to enact.

4.1 When to Trigger Adaptations?

An important aspect to consider when dealing with self-tuning of systems is determining *when* to trigger an adaptation. This aspect gains a paramount importance in DTMs, in particular when performing elastic scaling, replication switching or change in the replication degree. In fact, global reconfigurations and data migration can pose significant overhead on transactions processing, which may severely hinder performance during a non-negligible time window [62].

In this context, a key classification of existing self-tuning techniques is whether they react to workload changes, or they try to anticipate them. Another fundamental problem is related to the issue of distinguishing in a robust way actual workload changes from transient noise, which frequently affect workload metrics measurements in large scale systems. Finally, another relevant issue, which is at the basis of proactive schemes, is how to predict future workload trends. In the following we provide an overview of the key methodologies/building blocks that are used to address these issues. It should be noted that the techniques described below can be employed in a broad range of self-tuning systems, and their applicability is not restricted to adaptive DTM platforms.

Before describing each of these techniques, it is worth noting that in a DTM environment a workload can be characterized using a multitude of metrics. Besides classical/general-purpose metrics, like transactions arrival rate and CPU/bandwidth demand to perform operations, the workload of a DTM can be characterized also using DTM-specific metrics, such as the ratio of read-only vs update transactions, the number of accessed data items per transaction, and the transaction conflict probability.

Reacting to vs Predicting Workload Changes. A key characteristic that allows for coarsely classifying existing self-tuning mechanisms is whether they rely on *reactive* vs *proactive* approaches. Reactive schemes evaluate the need for reconfiguration based on the current workload, whereas proactive self-tuning strategies attempt to anticipate the need for changing system's configuration by predicting future workload trends.

Since reactive schemes track variations of the workload based on recent observations,they typically allow the system to react promptly even to abrupt workload changes due to exogenous factors (like flash crowds [63]), which would be very hard, if not impossible, to predict using proactive schemes. However, given that the reconfiguration is carried out against the current workload, reactive schemes can yield sub-optimal performance during transitory phases, especially in case the adaptation phase incurs a non-negligible latency.

On the other hand, the pros of proactive strategies coincide with the cons of reactive ones. By anticipating the need for changing system's configuration, adaptations can be enacted before the occurrence of workload changes. As a result, proactive approaches can reduce the period of time during which suboptimal configurations are used. On the other hand, the effectiveness of proactive approaches is strongly dependent on the accuracy of the mechanisms that they adopt to predict future workload trends (which we will overview shortly). For this reason, proactive and reactive schemes are sometimes combined into hybrid schemes [63,64,45].

Robust Change Detection. Workload measurement, especially in complex distributed platforms like DTMs, are typically subject to non-negligible noises. Hence, the robustness of any self-tuning scheme is strongly affected by its ability to distinguish small workload fluctuations, e.g., due to short transitory phases or transient spikes, from actual workload shifts, i.e., transitions from one workload to a different, stable one. This is a fundamental requisite to enforce the system's stability, i.e., to avoid its continuous oscillation among different states, namely configurations, due to frequent re-adaptations triggered by unavoidable, fleeting workload's fluctuations.

A principled approach to tackle this issue is based on the idea of considering the workload as a generic signal. Filtering techniques [65] can, then, be applied in order to reduce/remove noise and extract statistically meaningful information. One of the simplest examples of a filter is the Moving Average (MA), in which, given a time window composed by t intervals, the value v at observation j is given by $v_j = \sum_{i=j-t+1}^{j} \frac{v_i}{t}$; in the Exponential Moving Average (EMA), elements in the summation are given a weight that decreases as the measurement becomes older, in order to give more importance to recent measurements.

A more advanced filter employed to perform measurements in presence of noise is the Kalman Filter [66], which computes the value of the target metric as a weighted sum of the last prediction and the latest measurement. The weights reflect the confidence of such estimate and measurement and it is inversely proportional to the variance associated with those two values. The Kalman Filter represents a reference technique to track systems' parameters [67] and have been successfully applied in a wide range of applications, from CPU provisioning in virtualized environments [68] to performance optimization with energy constraints [69].

Another prominent related technique, originally introduced in the literature on statistical process control [70] to verify whether a process complies to its behavioral expectations, is the CUSUM (Cumulative Sum Control Chart) [71]. CUSUM involves the computation of a cumulative sum: noting x_n the n-th measurement for the target metric and w_n the corresponding weight, the cumulative sum at the n-th step, namely S_n, is expressed as $S_n = max\{0, S_{n-1} + w_n x_n\}$, with $S_0 = 0$. When S_n grows over a predefined threshold, a change in the metric is identified.

The CUSUM technique, whose employment has been borrowed from the manufacturing field, has been applied not only to workload monitoring and characterization for distributed transactional platforms [72], but also to tackle other issues like tracking faults in distributed systems [73] and detecting divergence from a desired QoS [74].

Workload Forecasting. As already mentioned, workload forecasting is a key problem at the basis of proactive self-tuning techniques. The techniques used to this purpose are typically borrowed from the literature on time-series analysis and forecasting, and can be classified depending on whether they operate in the time or in the frequency domain [75].
Time-domain methods. Techniques belonging to this category forecast the value for a metric in the next time window based on the raw measurements of such metric in the past. Auto Regression and Moving Averages methods are at the basis of a broad family of time-domain solutions: ARMA (Auto-Regressive Moving Average), which combines the two; ARIMA (AR Integrated MA), which generalizes the previous one to the case of non-stationary time series (i.e., time series whose shape changes over time); SARIMA (Seasonal ARIMA), which allows the ARIMA technique to incorporate preexistent knowledge about seasonal, namely recurring, behaviors [76]. Other popular solutions are based on the use of filtering techniques, such as the aforementioned Kalman Filter. In fact, due to its recursive nature, once instantiated, the Kalman Filter can be queried not only to filter out noisy components from the current measurements, but also to predict future values of the tracked workload metrics.

Frequency-domain methods. Techniques belonging to this category are aimed at extracting from time series information about seasonality and recurrence. Frequency-domain methods rely either on spectral analysis or on wavelet analysis. They are both based on the idea of decomposing a time series into a summation in the frequency domain: the former uses sinusoids as basis, the latter uses wavelets [76].

4.2 Which Adaptation to Trigger?

Once workload changes are detected, self-tuning systems need to decide which adaptation to trigger, if any, to react to such change. The identification of the optimal configuration is typically performed by means of performance models,

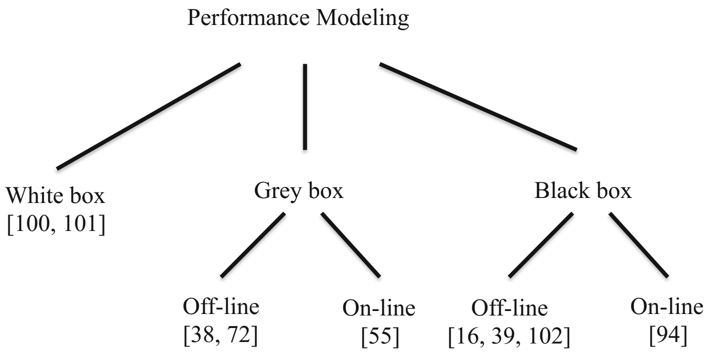

Fig. 3. Taxonomy of performance modeling techniques

which allow for the estimation/prediction of the system's performance in the various available configurations. The literature on performance modeling of computing systems is very prolific, and the models used in self-tuning system differ significantly in their nature and complexity. In Figure 3, we classify them into white, black and gray (an hybrid of black and white) box techniques, according to whether (and how) they exploit knowledge on the internal dynamics of the system. Moreover, we further classify black box, and hence grey box, approaches into off-line and on-line, depending on whether the model is built before putting the application in execution or at runtime.

White Box Modeling. This approach leverages on available expertise on the internal dynamics of systems and/or applications, and uses such knowledge to build an Analytical Model (AM) (e.g., based on queueing theory) or simulators, aimed at capturing how system's configuration and workload's parameters map onto performance [77]. Once defined, analytical models typically require no training (or a minimal profiling to obtain the value for some basic parameters) for being instantiated. In order to ensure their mathematical tractability, however, analytical models typically rely on approximations and simplifying assumptions on how the modeled system and/or its workload behave. Their accuracy can hence be challenged in scenarios (i.e., areas of the configurations' space or specific workload conditions) in which such approximations are too coarse, or are simply not matched. In addition, aside from possible re-evaluations of internal parameters, analytical models' inaccuracies are not amendable, as the mathematical characterization of the system's dynamics in encoded by means of immutable equations.

Black Box Modeling. This approach lies on the opposite side of the spectrum with respect to the white box solutions. Black box modeling does not require any knowledge about the target system/application's internal behavior. Conversely, it relies on a *training phase*, namely on observing the system's actual behavior under different configurations and while subject to different workloads, in order to infer a statistical performance model via different Machine Learning (ML) techniques [78]. Over the last years, these approaches have become more and more popular as tools for performance prediction of modern systems and applications, whose ever growing complexity challenges the viability of developing sufficiently detailed, and hence accurate, analytical models.

In practice, the accuracy achievable by black box models strongly depends on the representativeness of configurations and workloads that the ML has witnessed with during its training phase. This results in the ability of black box models to achieve a very good accuracy for scenarios sufficiently close to the ones observed during the training phase; on the other hand, predictions' accuracy of ML techniques is typically poor in regions of the parameters' space that were not sufficiently sampled during the training (in which case the model is often said to be used in extrapolation).

Unfortunately, the space of all possible configurations for a target system/application grows exponentially with the number of variables (a.k.a. features in the ML terminology) that can affect its performance — the so called curse of dimensionality [79]. Hence, in complex systems, like DTMs, the cost of conducting an exhaustive training process, spanning all possible configurations of the design and configuration's space and experimenting with all possible workloads, can typically be prohibitive.

Grey Box Modeling. Grey box approaches, as the name suggests, employ white and black model methodologies in hybrid fashions, so as to inherit the best features of the two worlds: the good accuracy in extrapolation (i.e., for unseen configuration/workloads) and minimal training time typical of white box models, and the robustness and possibility to incrementally enhance accuracy, via periodic retraining, of black box models.

Grey box techniques can, in their turn, be grouped into three categories.

- *Parameter fitting*: this solution relies on fitting techniques [80] to identify the values of (a subset of) the input parameters of a white box model, whose direct measurement is undesirable or infeasible. This is the case, for instance, of models that require detailed workload characterization [42] or service demand times [41], and whose measurement from an operational system may introduce prohibitive overheads. This technique is used also in case some parameters of white-box models do not map directly to any physical aspect of the system, and are instead used to encapsulate complex systems' dynamics that would be otherwise hard to capture explicitly via analytical techniques [58]. In these situations, fitting techniques can be used to determine the values of the unknown parameters that minimize the model's prediction errors over a given training set.
- *Divide et impera*: this technique consists in building performance models of individual parts of the entire system, which are either based on AM or on ML. The sub-models are then combined in order to obtain a prediction of the system as a whole [72,38]. This approach is particularly suited for

scenarios in which the internals dynamics of certain sub-components of the system are not known and/or are not easy to model using white-box analytical models, e.g., the networking infrastructure in a cloud-based distributed platform. The performance of these sub-components can then be predicted using black-box ML-based techniques, whereas white-box modeling can be used for the remainder of the system. By narrowing the domain over which ML techniques are used, their learning time is normally significantly reduced; also, the joint usage of white box models allows for achieving better accuracy in extrapolation when compared with pure black-box approaches.

- *Bootstrapping*: this methodology relies on an AM predictor to generate an initial synthetic training set for the ML, with the purpose of avoiding the initial, long profiling phase of the target application under different settings. Then, the ML is retrained over time in order to incorporate the knowledge coming from samples collected from the operational system [59,55].

While white box modeling is an inherently off-line technique, ML solutions, at the basis of purely black or grey box models, can be instantiated either off-line or on-line.

Off-line Learning. Off-line black box performance models are typically built by means of *Supervised Learning (SL)*, in which the ML algorithm is trained on labeled features, i.e., input for which the output is known.

In SL, the training algorithm, noted γ, is a function defined over the *training set* $D_{tr} = \{< \mathbf{x}, y >\}$, where $\mathbf{x} =< x_1, \ldots, x_n >$ is a point in a $n-$dimensional features' space, noted F, and y is the value of some unknown function $\phi : F \rightarrow C$. The co-domain C of the function may be a discrete set, whose elements are called *classes*, or a continuous space. The problem of learning the mapping of elements of F to C is called *classification* in the first case, and *regression* in the second one.

The output of γ is a function, also called model, noted Γ, which represents an approximation of ϕ over the features' space F. More precisely, a model $\Gamma : F \rightarrow C$ takes as input a point $\mathbf{x} \in F$, possibly not observed in D_{tr}, and returns a value $\hat{y} \in C$.

In off-line SL, the training set D_{tr} is assumed fully available to the learning algorithm. When new data is available, e.g. by gathering new sample from a running application, a new model can be built from scratch, considering the whole available training data set. Note that this palingenesis of the statistical model does not qualify as an instance of on-line learning, as we shall discuss briefly, as the model is built *ex novo* over an ever-increasing training set. Examples of off-line SL algorithms are Decision Trees, Support Vector Machines and Artificial Neural Networks [78].

On-line Learning. We distinguish three main approaches to on-line black box learning. The first one consists in on-line SL, according to which the model is built incrementally over a stream of training samples, i.e., only a subset of D_{tr} (possibly of cardinality 1) is available at the time, and it has to be incorporated in the model without being stored for further consideration [81]. Approaches in this domain typically assume that the learning algorithm can access each sample only once during the training phase. As a consequence, they normally require considerably less computational resources than off-line techniques, but may also achieve lower prediction accuracy.

A second on-line ML technique is *Reinforcement Learning (RL)*. RL aims at inferring the best way of performing actions in an environment (characterized in DTM context by a set of workload and performance indicators) given a state (i.e., a workload), so as to maximize some notion of cumulative reward (e.g., throughput). The main challenge tackled by RL techniques [82,83] is finding a balance between exploration (of untested actions for a given state) and exploitation (of available, and typically incomplete, knowledge), while minimizing the, so called, *regret*, that is the cumulative error with respect to the optimal strategy. Frequent explorations allow for acquiring a good knowledge of the rewards corresponding to different actions in a given state, but also causes the system to oscillate among several sub-optimal configurations, yielding to instability and hindering performance. On the other hand, an overly conservative policy, which does not test the available options sufficiently often, may get stuck in local maxima, especially in scenarios in which the reward distribution is subject to large variance (and may hence require a relatively large number of samples to be accurately estimated).

Finally, on-line black box self-tuning schemes can be based on optimization techniques like Gradient Descent or Genetic algorithms [84]. These approaches seek to minimize/maximize a given application's performance indicator: similarly to RL approaches, they combine exploration and exploitation; however, they do not encompass the notion of cumulative reward, thus differing from RL in the way the search of the optimal configuration is carried out and in the amount of information maintained about the system/application's state and previously performed explorations.

5 Elastic Scaling in DTM Systems

In this section we review solutions aimed at self-tuning the scale of DTMs. Though we focus on this kind of platform, we also include in the analysis solutions that have been proposed and evaluated in the broader field of elastic scaling of distributed data platforms and which could be applied also to the case of DTMs.

In our analysis we will focus on three main technical challenges, which need to be tackled in order to implement effective elastic scaling solutions for DTM, namely: how to preserve consistency during elastic scaling, when to trigger elastic scaling, how to determine the new scale of the DTM.

How Is Elastic Scaling Supported? DTM can either scale vertically, namely, by changing the number of concurrent threads active in each of the platform's nodes, or horizontally, namely, changing the number of nodes in the platform. In the first case, the scaling procedure does not encompass inter-node synchronization or state transfer, as it simply consists of activating/deactivating the desired number of threads [57].

Scaling out a DTM is, conversely, a much more challenging task given the stateful nature of the platform that implies the need for a state transfer phase and the constraint of preserving the consistent and atomic access to data items during the reconfiguration. In this paragraph we survey some state transfer techniques that have been proposed to elastically scale databases but that are applicable also to the case of DTMs.

The simplest solution to scale out a distributed transactional platform is the *stop and go* technique, which naively consists in blocking transactions execution during the state transfer and restoring it when it is over. Of course, the major drawback of this solution is that it implies service unavailability during the scaling phase, and it is, thus, employed only when there is no other option available [21].

For this reason, a number of solutions have been proposed to perform the state transfer at the application level, i.e., relying only on the transactional middleware of the platform.

A first one represents an improvement over the *s*top and go: while a new node is being initiated it cannot serve requests, but other nodes can, thus maintaining the service available. This technique basically consists of three phases. In the first one, a new node is spawned and starts receiving data from the source nodes designated by the data placement component. In the second one, it receives newer versions of data that it has already received during the first phase, but which have been updated in the meanwhile. In the last phase, the new node receives the last stream of data and starts processing transactions; in order to allow the new node to catch up with the state of running nodes without breaking atomicity and consistency, this phase may require all the nodes in the system to stop processing transactions, thus resulting into a short service unavailability window. This technique has been applied to the context of live migrations of databases in multi-tenant [85] and single instance [86,46] environments. Optimized variants for partially replicated systems also exist, in which the amount of data sent by live nodes to the joining one(s) is evenly split, thus resulting into an optimal load balancing [87].

A further optimization of the aforementioned scheme consists in allowing the new node to start serving transactions as soon as it gets data. In order to maintain atomic and consistent access to data, schemes relying on this optimization integrate the state transfer with the distribution and concurrency control protocol employed by the platform [88].

Another technique employed for elastic scaling of distributed databases, especially in the case of multi-tenant infrastructures, consists in transferring a snapshot of the database, i.e., an image of the database state at a given point in time. This can be achieved by means of VM migration [89] and backup tools [90,91] or by relying on the presence of a Network Attached Storage [85].

Finally, Barker et al. [92] show that no-one-size-fits-all solutions exists in the landscape of the described techniques for databases migration and state transfer.

Therefore, they introduce a hybrid scheme that automatically selects the best elastic scaling scheme to employ, choosing between a black-box VM migration and a database-aware, application-level state transfer.

When to Trigger Elastic Scaling? As introduced in Section 4, the literature on elastic scaling of distributed data platforms includes proposals based on the reactive and proactive approaches.

Among the solutions based on reactive schemes, Exponential Moving Average (EMA) is employed in the provisioning of a one-copy serializable database by Soundararajan et Amza [46] and of an eventually consistent data store by Trushkowsky et al. [93]: given a current raw measurement v_r and the output of last EMA computation v_l, the current value for target metric v_c (average response time of queries in the first case and arrival rate to a dataset partition in the second one) is obtained as $v_c = \alpha v_r + (1 - \alpha)v_l$. Here, α is a weighting factor: the higher, the faster older observation are discounted.

Scaling the size of a DTM, however, is a very onerous operation, as it triggers a state transfer phase that can induce significant additional load on the system for a potentially long time [62,92,89]. Thus, as a result of relying on a reactive scheme to trigger the elastic scaling, during the whole reconfiguration phase, the platform can suffer from severe performance degradation due to a sub-optimal configuration with respect to the incoming workload. To avoid such a shortcoming, the majority of recent research works on automatic resource provisioning rely on proactive schemes to trigger the elastic scaling of data platforms.

Approaches operating in the time domain, based on simple linear extrapolation [94] and filtering [62], have been applied to drive the elastic scaling of distributed databases. Solutions relying on time series analysis, namely ARMA and ARIMA, have also been frequently applied to drive automatic elastic scaling policies for Cloud applications [43,95,44].

Likewise, works based on time series analysis in the frequency domain find application in automatic resource provisioning scheme for Cloud infrastructures. They are either used alone, as in the case of the Agile system [96], or in conjunction with ANN in a recent work by Napoli et al. [97].

Approaches [45,64] combining reactive and proactive techniques, especially in QoS-oriented and SLA-based Cloud platforms, typically favor a more aggressive scheme in adding nodes and more conservative ones in scaling down removing nodes from a platform. The rationale behind this choice is that the cost, both monetary and in terms of performance, for maintaining resources that are not strictly necessary to guarantee a desired QoS is lower than the one resulting from an unfortunate scaling down choice, both because of the overhead due to a new scaling up phase and to the penalties stemming from possible SLA violations.

In the Cloud-TM data platform [40,98], Kalman filter and polynomial regression are employed to predict future workloads; however, they are complemented by a reactive scheme based on a filter that detects variations of average values over two consecutive time-windows, and the CUSUM algorithm. Different trade-offs between pro and reactiveness can be achieved depending on the parametrization of such algorithms. A similar approaches is undertaken also in ShuttleDB [92], where a threshold-based reactive scheme is complemented by times series forecasting by means of an ARIMA model. Iqbal et al. [45] propose a hybrid scheme which is reactive in acquiring resources, while it employs a second order regression to detect over-provisioning with respect to the incoming workload and, accordingly, release resources. In MeT [64], resources are greedily acquired in a non-linear and iterative fashion, i.e., if the system is under-provisioned, the number of acquired nodes at iteration i is twice as much as at last iteration; nodes in the system are, instead, released linearly, namely, one by one. Ali-Eldin et al. [63] provide a thorough analysis of controllers for elastic Cloud-based application relying on nine different schemes combining reactive and proactive approaches. Their work suggest that, indeed, hybrid schemes do perform better than pure ones.

With the exception of the techniques integrated in the Cloud-TM platform, the aforementioned solutions typically target either stateless/non-transactional platforms or transactional ones with external storage systems (e.g., Network Attached Storages) or backup services.

Their application to DTMs without those specific supports or in typical, commercial Cloud deployment is, hence, not straightforward. Moreover, such proposals do not account for other potential concurrent reconfigurations of the platforms at other levels, e.g., at the consistency protocol one. Challenging research problems in this direction that demand further investigation are the estimation of the duration of the reconfiguration phase and of SLA violations incurred during that time.

Which Scale to Choose? A plethora of analytical and simulative models for distributed transactional data platforms exist [99,100] that are aimed at computing the performance of the platform when deployed over different number of nodes. However, they mainly target relational databases and do not encompass complex dynamics that stem from elastically scaling the platform at runtime, like the variation in data locality. For this reason, in recent years, performance modeling and forecasting specifically aimed at supporting elastic scaling of DTM has garnered much attention, resulting into solutions that cover the whole spectrum of the techniques introduced in Section 4.2.

A pure white box model, relying on Parallel Discrete Event Simulation, has been proposed by Di Sanzo et al. [101]. It allows for the definition of trace based workloads in order to forecast the effect of elastically scaling, both vertically and horizontally, a DTM, encompassing generic data placement schemes and arbitrary data access patterns exhibited by the hosted application.

Pure black box approaches, instead, have been undertaken in [50,102], where ANN are employed to predict transactions' throughput and response time while varying the number of nodes composing a DTM. In particular, the work in [102] allows for supporting what-if analysis at the granularity of individual transactional classes, and not only on the overall average performance of the entire transactional workload.

A *divide et impera* grey box modeling approach is proposed by Didona et al., which targets performance prediction of fully [72] and partially replicated [38] DTMs when varying its scale over Cloud infrastructures. In such approach, analytical modeling is employed to model resource contention over the CPU and to capture transactions' conflict probability on data. Conversely, ML, and specifically decision tree based regression, is employed to predict the latency of network-bound operations, e.g., the retrieval of remote data and the execution of the distributed commit phase.

A variant of the *bootstrapping* grey box methodology is proposed in [57], and extended in [103], with the aim of determining the scale for a DTM application that results in the higher throughput. This approach combines analytical modeling, supervised learning and pure exploration in order to build a performance model that incrementally enhances its accuracy. A DT regressor is employed to learn at runtime a corrective function to be applied to the output of the base performance predictor (based on [72]) so as to progressively reduce its prediction error. The DT is incrementally trained over the base model's mis-predictions for workloads and scales that the DTM has experienced with. In order to widen the training set of the DT without incurring the cost of state transfer, different levels of MPL are explored for a given workload and number of nodes in the DTM.

6 Adaptation of the Data Consistency Protocol

In this section we review the most relevant solutions that focus on the adaptation of the protocol used to enforce data consistency in DTM platforms. Each system is described according to the three major concerns for supporting automatic protocol switching in DTM platforms: how is consistency ensured despite the on-line switching between different data consistency protocols, when the system should switch the protocol, and which is the most suitable consistency protocol according to the current conditions.

How Is Protocol Switching Supported? There are two main architectural approaches for protocol switching in DTM platforms, ad-hoc and generic, which explore different trade-offs between simplicity, efficiency and generality.

In the ad-hoc approach, the system is designed to accommodate specific and predetermined protocols and it is highly tailored to provide seamless switching mechanisms between protocols, i.e., to minimize the impact on performance during the switching phase. By exploiting the knowledge on the internal dynamics of the origin and target consistency protocols (for instance, how they are implemented), one can indeed design specialized switching mechanisms that exploit possible compatibilities with the purpose of reducing the overhead and/or duration of the switching phase. Typically, it is not possible to support the switching from/to additional protocols without making profound changes in the system.

Examples of these systems include PolyCert [16] and HTR [26]. PolyCert is a DTM that relies on three certification-based consistency protocols: non-voting certification, which sends the read-set of transactions as is; Bloom filter certification, which encodes the transaction's read-set in a Bloom filter, minimizing the size of the messages exchanged by nodes but increasing the complexity of processing the received message; and voting certification, in which only the write-set of transactions is disseminated but replicas must wait for a commit decision from the node where the transaction originally executed. As transactions finish their local execution, the protocol that minimizes the commit phase is selected from the three available (using techniques described further ahead in the section), improving therefore the throughput of the system. HTR also determines the optimal protocol on a per transaction basis: based on the abort rate on the moment each transaction is issued, either the deferred update model, which takes advantage of multicore hardware to process transactions in parallel, is chosen or the state machine approach, which guarantees an abort free execution. Both systems are tailored for those specific protocols and do not contemplate the addition of others.

Ideally, developers should be allowed to choose the most suitable replication protocols for their systems and workloads. Also, these protocols should be easy to plug into the system, and oblivious of other protocols (i.e., there should be no dependencies between protocols neither while the system is in normal operation nor when during the switching phase).

Recently, a new approach was proposed that offers both flexibility and performance. MorphR [39] is a framework that supports multiple replication protocols by only requiring their adherence to a specified API. It provides two mechanisms for the switching phase: stop and go and fast switching. The first approach relies on a blocking scheme to guarantee that there is no transaction from the old protocol running in the system when the new protocol starts executing, ensuring isolation between the switching protocols and avoiding the need to implement interactions between protocols. The second approach leverages on the knowledge of developers to implement specialized switching algorithms between pairs of protocols enabling their co-existence so that the performance of the system is not affected by this adaptation. MorphR's prototype was tested with three very different protocols representing distinct classes of replication approaches: 2PC, PB and a TOB-based scheme.

When to Switch? The most common approach to trigger switching in these systems is employing reactive schemes, that detect changes in the workload and react to those changes. Most adaptive DTM systems [39,72] rely on this approach, especially systems like HTR and PolyCert, which determine the best protocol on a per-transaction basis and transactions' operations are not known prior to their actual execution.

On the opposite side of the spectrum, CloudTM platform [40] integrates workload and resource demand prediction schemes, by including algorithms for time-series forecasting which allow predicting future workload's trends and allow the system to enact proactive self-tuning schemes. This functionality represents a fundamental building block for any proactive adaptation scheme, i.e., schemes triggering reconfigurations of the platform anticipating imminent workload's changes, which are particularly desirable in case the platform's reconfiguration (as in the case of elastic scaling) can have non-negligible latencies.

Which Protocol to Choose? The most straightforward way to approach the problem of determining the most suitable protocol is to set thresholds that, using one or more metrics, define the scenarios in which each protocol delivers (or is expected to deliver) the best performance. HTR follows this approach: it monitors the abort rate of the system before each transaction and if it exceeds a certain threshold, the transaction is executed in the state machine mode, which guarantees abort free execution. When the abort rate is lower than the set threshold, transactions will revert to executing in the deferred update mode.

However, threshold-based approaches become very hard to properly tune when the complexity of the replication schemes and workloads increases, as the increasing number of metrics and thresholds will eventually become unmanageable by an administrator. Let aside, the lack of flexibility imposed by the usage of fixed values for the thresholds. Both PolyCert and MorphR rely on the black box approach, namely machine learning techniques which were previously presented in Section 4, to cope with a larger number of protocols, with potentially complex algorithms, system configurations and workloads. While PolyCert assesses protocol suitability on a per transaction basis (i.e., each transaction issued will be certified with the protocol that minimizes its total execution time), MorphR evaluates the state of the system periodically (at a frequency tuned by the administrator) to verify whether the protocol in use is the optimal one and, if not, changes the protocol used by the entire system to match the most suitable option for the observed conditions.

However, a pure black box approach will not be able to cope with workloads and system configurations that were not included in the data used as its training set. The grey box approach, used in TAS [72,38], relies on analytical models designed to predict the behavior of 2PC and PB regardless of the workload and system configuration (number of machines, hardware used, etc.). This method is especially well tailored for systems in which administrators do not have prior knowledge of workloads and deployment configurations or when these two aspects are constantly varying. On the other hand, taking advantage of this approach entails possessing a very deep knowledge of the system's internals to be able to design a complete and accurate model.

7 Conclusions and Open Research Questions

In this chapter we have investigated the problem of designing self-tuning DTM platforms. Along the way, we have exposed some of the key trade-offs in the design of the main components of DTM systems, and recalled some of the base methodologies that are commonly employed in self-tuning systems. We have then focused our attention on two specific self-tuning problems, elastic scaling and adaptation of the distributed consistency protocol, and critically analyzed existing literature in these areas.

The analysis that we have conducted in this chapter shows that, despite being a relatively young research area, the existing literature encompasses already a number of self-tuning solutions that target the key building blocks of DTM platforms. On the other hand, our analysis suggests also that there are still a number of unexplored areas and open research problems, which represent interesting opportunities for future research.

In the elastic scaling area, for instance, we are not aware of solutions for estimating the impact on performance due to the occurrence of the state transfer activities that are necessary to redistribute data across nodes of the DTM platform. Another aspect that has not been satisfactorily addressed, to the best of our knowledge, by existing solutions in the area of elastic scaling of DTM is the prediction of the locality shifts (i.e., the change in the probability of incurring in remote accesses) due to the redistribution of data among the nodes caused by the elastic scaling process.

As for the dynamic switching of the DTM consistency protocol, existing solutions only take into account adaptations of the distributed consistency mechanisms, and do not seek integration with the self-tuning mechanisms for non-distributed TMs (e.g., targeting the local concurrency control or the thread mapping).

A related, albeit more fundamental open question, is how to effectively integrate the various self-tuning mechanisms proposed in literature and targeting different modules/parameters of DTM platforms. These systems are constituted by a complex ecosystem of components, each one associated with specific key performance indicators, utility functions and monitorable/tunable parameters. These components exhibit non-trivial mutual interdependencies; hence, in general, it is not possible to optimize separately different modules of a DTM, as the effect on performance of tuning different parameters are often intertwined. The complexity of this type of system is simply too high for monolithic self-tuning approaches, i.e., approaches that try to optimize the system as a whole by trying to identify all possible relations among the feasible adaptation alternatives of the entire ecosystem of components. Alternative, modular approaches would be highly desirable, as they would allow for unifying the large set of existing self-tuning mechanisms that target different aspects of DTMs. To the best of our knowledge, this problem is still unexplored by existing research.

References

1. Yoo, R.M., Hughes, C.J., Lai, K., Rajwar, R.: Performance evaluation of Intel® transactional synchronization extensions for high-performance computing. In: International Conference for High Performance Computing, Networking, Storage and Analysis, pp. 1–19. ACM (2013)
2. Jacobi, C., Slegel, T., Greiner, D.: Transactional memory architecture and implementation for ibm system z. In: Proceedings of the Annual nternational Symposium on Microarchitecture (MICRO), pp. 25–36. IEEE Computer Society (2012)
3. Herlihy, M.P., Sun, Y.: Distributed transactional memory for metric-space networks. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 324–338. Springer, Heidelberg (2005)
4. Romano, P., Carvalho, N., Rodrigues, L.: Towards distributed software transactional memory systems. In: Proceedings of the Workshop on Large-Scale Distributed Systems and Middleware (LADIS), pp. 1–4. ACM (2008)
5. Bocchino, R.L., Adve, V.S., Chamberlain, B.L.: Software transactional memory for large scale clusters. In: Proceedings of the Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 247–258. ACM (2008)

6. Romano, P., Rodrigues, L., Carvalho, N., Cachopo, J.: Cloud-tm: harnessing the cloud with distributed transactional memories. SIGOPS Operating Systems Review 44, 1–6 (2010)
7. Couceiro, M., Romano, P., Carvalho, N., Rodrigues, L.: D2STM: Dependable distributed software transactional memory. In: Proceedings of the Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 307–313. IEEE Computer Society (2009)
8. Palmieri, R., Quaglia, F., Romano, P.: Aggro: Boosting stm replication via aggressively optimistic transaction processing. In: Proceedings of the International Symposium on Network Computing and Applications (NCA), pp. 20–27. IEEE Computer Society (2010)
9. Carvalho, N., Romano, P., Rodrigues, L.: Asynchronous lease-based replication of software transactional memory. In: Gupta, I., Mascolo, C. (eds.) Middleware 2010. LNCS, vol. 6452, pp. 376–396. Springer, Heidelberg (2010)
10. Hendler, D., Naiman, A., Peluso, S., Quaglia, F., Romano, P., Suissa, A.: Exploiting locality in lease-based replicated transactional memory via task migration. In: Afek, Y. (ed.) DISC 2013. LNCS, vol. 8205, pp. 121–133. Springer, Heidelberg (2013)
11. Fernandes, S.M., Cachopo, J.A.: Strict serializability is harmless: A new architecture for enterprise applications. In: Proceedings of International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (SPLASH), pp. 257–276. ACM (2011)
12. Peluso, S., Ruivo, P., Romano, P., Quaglia, F., Rodrigues, L.: When scalability meets consistency: Genuine multiversion update-serializable partial data replication. In: International Conference on Distributed Computing Systems (ICDCS), pp. 455–465. IEEE (2012)
13. Peluso, S., Romano, P., Quaglia, F.: SCORe: A scalable one-copy serializable partial replication protocol. In: Narasimhan, P., Triantafillou, P. (eds.) Middleware 2012. LNCS, vol. 7662, pp. 456–475. Springer, Heidelberg (2012)
14. Schneider, F.B.: Replication management using the state-machine approach. ACM Press/Addison-Wesley Publishing Co. (1993)
15. Pedone, F., Guerraoui, R., Schiper, A.: The database state machine approach. Distributed Parallel Databases 14(1), 71–98 (2003)
16. Couceiro, M., Romano, P., Rodrigues, L.: PolyCert: Polymorphic self-optimizing replication for in-memory transactional grids. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 309–328. Springer, Heidelberg (2011)
17. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: Proceedings of the SIGMOD International Conference on Management of Data, pp. 173–182. ACM (1996)
18. Kemme, B., Pedone, F., Alonso, G., Schiper, A., Wiesmann, M.: Using optimistic atomic broadcast in transaction processing systems. IEEE Transactions on Knowledge and Data Engineering (TKDE) 15(4), 1018–1032 (2003)
19. Ruivo, P., Couceiro, M., Romano, P., Rodrigues, L.: Exploiting total order multicast in weakly consistent transactional caches. In: Proceedings of the Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 99–108. IEEE Computer Society (2011)
20. Paiva, J., Ruivo, P., Romano, P., Rodrigues, L.: Autoplacer: Scalable self-tuning data placement in distributed key-value stores. In: Proceedings of the International Conference on Autonomic Computing (ICAC), pp. 119–131. USENIX, San Jose (2013)
21. Marchioni, F., Surtani, M.: Infinispan Data Grid Platform. Packt Publishing (2012)
22. Dash, A., Demsky, B.: Integrating caching and prefetching mechanisms in a distributed transactional memory. IEEE Transactions on Parallel and Distributed Systems (TPDS) 22(8), 1284–1298 (2011)

23. Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., Lewin, D.: Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: Proceedings of the Symposium on Theory of Computing (STOC), pp. 654–663. ACM (1997)
24. Demmer, M.J., Herlihy, M.P.: The arrow distributed directory protocol. In: Kutten, S. (ed.) DISC 1998. LNCS, vol. 1499, pp. 119–133. Springer, Heidelberg (1998)
25. Herlihy, M.P., Sun, Y.: Distributed transactional memory for metric-space networks. In: Fraigniaud, P. (ed.) DISC 2005. LNCS, vol. 3724, pp. 324–338. Springer, Heidelberg (2005)
26. Kobus, T., Kokocinski, M., Wojciechowski, P.T.: Hybrid replication: State-machine-based and deferred-update replication schemes combined. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS), pp. 286–296. IEEE (2013)
27. Carvalho, N., Romano, P., Rodrigues, L.: A generic framework for replicated software transactional memories. In: Proceedings of the International Symposium on Networking Computing and Applications (NCA), pp. 271–274. IEEE Computer Society (2011)
28. Pimentel, H., Romano, P., Peluso, S., Ruivo, P.: Enhancing locality via caching in the gmu protocol. In: Proceedings of the International Symposium on Cluster, Cloud and Grid Computing (CCGRID). IEEE Computer Society (2014)
29. Sovran, Y., Power, R., Aguilera, M.K., Li, J.: Transactional storage for geo-replicated systems. In: Proceedings of the Symposium on Operating Systems Principles (SOSP), pp. 385–400. ACM (2011)
30. Kemme, B., Alonso, G.: A suite of database replication protocols based on group communication primitives. In: Proceedings of the International Conference on Distributed Computing Systems (ICDCS), pp. 156–163. IEEE Computer Society (1998)
31. Patiño-Martínez, M., Jiménez-Peris, R., Kemme, B., Alonso, G.: Scalable replication in database clusters. In: Herlihy, M.P. (ed.) DISC 2000. LNCS, vol. 1914, pp. 315–329. Springer, Heidelberg (2000)
32. Manassiev, K., Mihailescu, M., Amza, C.: Exploiting distributed version concurrency in a transactional memory cluster. In: Proceedings of the Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 198–208. ACM (2006)
33. Franklin, M.J., Carey, M.J., Livny, M.: Transactional client-server cache consistency: Alternatives and performance. ACM Transactions on Database Systems (TODS) 22(3), 315–363 (1997)
34. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Computing Surveys (CSUR) 36(4), 372–421 (2004)
35. Lamport, L.: The part-time parliament. ACM Transactions on Computing Systems (TOCS) 16(2), 133–169 (1998)
36. Agrawal, D., Alonso, G., El Abbadi, A., Stanoi, I.: Exploiting atomic broadcast in replicated databases (extended abstract). In: Lengauer, C., Griebl, M., Gorlatch, S. (eds.) Euro-Par 1997. LNCS, vol. 1300, pp. 496–503. Springer, Heidelberg (1997)
37. Muñoz-Escoí, F.D., Irún-Briz, L., Galdámez, P., Decker, H., Bernabéu, J., Bataller, J., del Carmen Bañuls, M.: Globdata: A platform for supporting multiple consistency modes. In: Proceedings of the International Conference on Information Systems and Databases (ISDB), pp. 104–109. Acta Press (2002)
38. Didona, D., Romano, P.: Performance modelling of partially replicated in-memory transactional stores. In: Proceedings of the International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). IEEE (2014)

39. Couceiro, M., Ruivo, P., Romano, P., Rodrigues, L.: Chasing the optimum in replicated in-memory transactional platforms via protocol adaptation. In: Proceedings of the International Conference on Dependable Systems and Networks (DSN), pp. 1–12. IEEE Computer Society (2013)
40. Didona, D., Romano, P.: Self-tuning transactional data grids: The cloud-tm approach. In: Proceedings of the Symposium on Network Cloud Computing and Applications (NCCA), pp. 113–120. IEEE (2014)
41. Singh, R., Sharma, U., Cecchet, E., Shenoy, P.J.: Autonomic mix-aware provisioning for non-stationary data center workloads. In: Proceedings of the International Conference on Autonomic Computing (ICAC), pp. 21–30. ACM (2010)
42. Zhang, Q., Cherkasova, L., Mi, N., Smirni, E.: A regression-based analytic model for capacity planning of multi-tier applications. Cluster Computing 11(3), 197–211 (2008)
43. Roy, N., Dubey, A., Gokhale, A.S.: Efficient autoscaling in the cloud using predictive models for workload forecasting. In: Proceedings of the International Conference on Cloud Computing (CLOUD), pp. 500–507. IEEE (2011)
44. Chen, G., He, W., Liu, J., Nath, S., Rigas, L., Xiao, L., Zhao, F.: Energy-aware server provisioning and load dispatching for connection-intensive internet services. In: Symposium on Networked Systems Design & Implementation (NSDI), pp. 337–350. USENIX Association (2008)
45. Iqbal, W., Dailey, M.N., Carrera, D., Janecek, P.: Adaptive resource provisioning for read intensive multi-tier applications in the cloud. Future Generation Computing Systems 27(6), 871–879 (2011)
46. Soundararajan, G., Amza, C.: Reactive provisioning of backend databases in shared dynamic content server clusters. ACM Transactions on Adaptive and Autonomous Systems (TAAS) 1(2), 151–188 (2006)
47. You, G.-w., Hwang, S.-w., Jain, N.: Scalable load balancing in cluster storage systems. In: Kon, F., Kermarrec, A.-M. (eds.) Middleware 2011. LNCS, vol. 7049, pp. 101–122. Springer, Heidelberg (2011)
48. Curino, C., Jones, E., Zhang, Y., Madden, S.: Schism: A workload-driven approach to database replication and partitioning. Proceedings of the VLDB Endowment 3(1-2), 48–57 (2010)
49. Turcu, A., Palmieri, R., Ravindran, B.: Automated data partitioning for highly scalable and strongly consistent transactions. In: Proceedings of the International Systems and Storage Conference (SYSTOR), pp. 1–11. ACM (2014)
50. di Sanzo, P., Rughetti, D., Ciciani, B., Quaglia, F.: Auto-tuning of cloud-based in-memory transactional data grids via machine learning. In: Proceedings of the Symposium on Network Cloud Computing and Applications (NCCA), pp. 9–16. IEEE (2012)
51. Vale, T.M., Dias, R.J., Lourenço, J.M.: On the relevance of total-order broadcast implementations in replicated software transactional memories. In: Lourenço, J.M., Farchi, E. (eds.) MUSEPAT 2013 2013. LNCS, vol. 8063, pp. 49–60. Springer, Heidelberg (2013)
52. Mocito, J., Rodrigues, L.: Run-time switching between total order algorithms. In: Nagel, W.E., Walter, W.V., Lehner, W. (eds.) Euro-Par 2006. LNCS, vol. 4128, pp. 582–591. Springer, Heidelberg (2006)
53. Mocito, J., Rosa, L., Almeida, N., Miranda, H., Rodrigues, L., Lopes, A.: Context adaptation of the communication stack. International Journal of Parallel, Emergent and Distributed Systems 21(3), 169–181 (2006)
54. Didona, D., Carnevale, D., Galeani, S., Romano, P.: An extremum seeking algorithm for message batching in total order protocols. In: Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pp. 89–98. IEEE (2012)

55. Romano, P., Leonetti, M.: Self-tuning batching in total order broadcast protocols via analytical modelling and reinforcement learning. In: Proceedings of the International Conference on Computing, Networking and Communications, ICNC, pp. 786–792. IEEE (2011)
56. Wang, Q., Kulkarni, S., Cavazos, J., Spear, M.F.: A transactional memory with automatic performance tuning. ACM Transactions on Architecture and Code Optimization (TACO) 8(4), 1–54 (2012)
57. Didona, D., Felber, P., Harmanci, D., Romano, P., Schenker, J.: Identifying the optimal level of parallelism in transactional memory applications. Computing (2013)
58. di Sanzo, P., Re, F.D., Rughetti, D., Ciciani, B., Quaglia, F.: Regulating concurrency in software transactional memory: An effective model-based approach. In: Proceedings of the International Conference on Self-Adaptive and Self-Organizing Systems (SASO), pp. 31–40. IEEE (2013)
59. Rughetti, D., Di Sanzo, P., Ciciani, B., Quaglia, F.: Analytical/ml mixed approach for concurrency regulation in software transactional memory. In: Proceedings of the International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 81–91. IEEE (2014)
60. Felber, P., Fetzer, C., Riegel, T.: Dynamic performance tuning of word-based software transactional memory. In: Proceedings of the Symposium on Principles and Practice of Parallel Programming, pp. 237–246. ACM (2008)
61. Castro, M.B., Góes, L.F.W., Méhaut, J.F.: Adaptive thread mapping strategies for transactional memory applications. Journal of Parallel and Distributed Computing (JPDC) 74(8), 2845–2859 (2014)
62. Chen, J., Soundararajan, G., Amza, C.: Autonomic provisioning of backend databases in dynamic content web servers. In: Proceedings of the International Conference on Autonomic Computing (ICAC), pp. 231–242. IEEE (2006)
63. Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: Proceedings of the Network Operations and Management Symposium (NOMS), pp. 204–212. IEEE (2012)
64. Cruz, F., Maia, F., Matos, M., Oliveira, R., Paulo, J., Pereira, J., Vilaça, R.: Met: workload aware elasticity for nosql. In: Proceedings of EuroSys, pp. 183–196. ACM (2013)
65. Shenoi, B.A.: Introduction to Digital Signal Processing and Filter Design. John Wiley & Sons (2005)
66. Kalman, R.: A new approach to linear filtering and prediction problems. Journal of Basic Engineering 82, 35–45 (1960)
67. Zheng, T., Woodside, C.M., Litoiu, M.: Performance model estimation and tracking using optimal filters. IEEE Transactions on Software Engineering (TOSE) 34(3), 391–406 (2008)
68. Kalyvianaki, E., Charalambous, T., Hand, S.: Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In: Proceedings of the International Conference on Autonomic Computing (ICAC), pp. 117–126. IEEE (2009)
69. Hoffmann, H., Maggio, M.: Pcp: A generalized approach to optimizing performance under power constraints through resource management. In: Proceedings of the International Conference on Autonomic Computing (ICAC), pp. 241–247. USENIX Association (2014)
70. Wheeler, D.J.: Understanding Statistical Process Control, 3rd edn. SPC Press & Statistical Process Control, Inc. (2010)
71. Page, E.S.: Continuous inspection schemes. Biometrika 41(1), 100–115 (1954)
72. Didona, D., Romano, P., Peluso, S., Quaglia, F.: Transactional auto scaler: elastic scaling of replicated in-memory transactional data grids. ACM Transactions on Adaptive and Autonomous Systems (TAAS) 9(2) (July 2014)

73. Nguyen, H., Tan, Y., Gu, X.: Pal: Propagation-aware anomaly localization for cloud hosted distributed applications. In: Proceedings of Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques (SLAML), pp. 1–8. ACM (2011)
74. Amin, A., Colman, A., Grunske, L.: Statistical detection of qos violations based on cusum control charts. In: Proceedings of the International Conference on Performance Engineering (ICPE), pp. 97–108. ACM (2012)
75. Chatfield, C.: The analysis of time series: An introduction, 6th edn. CRC Press (2004)
76. Shumway, R.H., Stoffe, D.S.: Time Series Analysis and Its Applications, 3rd edn. Springer Texts in Statistics (2011)
77. Tay, Y.C.: Analytical Performance Modeling for Computer Systems, 2nd edn. Synthesis Lectures on Computer Science. Morgan & Claypool Publishers (2013)
78. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)
79. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
80. Marquardt, D.W.: An algorithm for least-squares estimation of nonlinear parameters. SIAM Journal on Applied Mathematics 11(2), 431–441 (1963)
81. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD), pp. 71–80. ACM (2000)
82. Auer, P.: Using upper confidence bounds for online learning. In: Proceedings of the Annual Symposium on Foundations of Computer Science (FOCS), pp. 270–279. IEEE Computer Society (2000)
83. Watkins, C.J.C.H., Dayan, P.: Technical note q-learning. Machine Learning 8, 279–292 (1992)
84. Russell, S.J., Norvig, P.: Artificial Intelligence - A Modern Approach, 2nd edn. Pearson Education (2010)
85. Das, S., Nishimura, S., Agrawal, D., El Abbadi, A.: Albatross: Lightweight elasticity in shared storage databases for the cloud using live data migration. PVLDB 4(8), 494–505 (2011)
86. Minhas, U.F., Liu, R., Aboulnaga, A., Salem, K., Ng, J., Robertson, S.: Elastic scale-out for partition-based database systems. In: ICDE Workshops, pp. 281–288 (2012)
87. Raghavan, N., Vitenberg, R.: Balancing the communication load of state transfer in replicated systems. In: International Symposium on Resliable Distributed Systems (SRDS), pp. 41–50. IEEE (2011)
88. Elmore, A.J., Das, S., Agrawal, D., El Abbadi, A.: Zephyr: live migration in shared nothing databases for elastic cloud platforms. In: Proceedings of the SIGMOD International Conference on Management of Data, pp. 301–312. ACM (2011)
89. Cecchet, E., Singh, R., Sharma, U., Shenoy, P.J.: Dolly: virtualization-driven database provisioning for the cloud. In: Proceedings of the International Conference on Virtual Execution Environments (VEE), pp. 51–62. ACM (2011)
90. Barker, S.K., Chi, Y., Moon, H.J., Hacigümüs, H., Shenoy, P.J.: "cut me some slack": latency-aware live migration for databases. In: International Conference on Extending Database Technology (EDBT), pp. 432–443. ACM (2012)
91. Sousa, F.R.C., Machado, J.C.: Towards elastic multi-tenant database replication with quality of service. In: Proceedings of the International Conference on Utility and Cloud Computing, pp. 168–175. IEEE (2012)
92. Barker, S., Chi, Y., Hacigümüs, H., Shenoy, P., Cecchet, E.: Shuttledb: Database-aware elasticity in the cloud. In: Proceedings of the International Conference on Autonomic Computing (ICAC), pp. 33–43. USENIX Association (2014)

93. Trushkowsky, B., Bodík, P., Fox, A., Franklin, M.J., Jordan, M.I., Patterson, D.A.: The scads director: Scaling a distributed storage system under stringent performance requirements. In: Proceedings of the Conference on File and Storage Technologies (FAST), pp. 163–176. USENIX Association (2011)
94. Ghanbari, S., Soundararajan, G., Chen, J., Amza, C.: Adaptive learning of metric correlations for temperature-aware database provisioning. In: Proceedings of the International Conference on Autonomic Computing (ICAC), pp. 1–26. IEEE (2007)
95. Chandra, A., Gong, W., Shenoy, P.J.: Dynamic resource allocation for shared data centers using online measurements. In: Proceedings of the International Conference on Measurements and Modeling of Computer Systems, pp. 300–301. ACM (2003)
96. Nguyen, H., Shen, Z., Gu, X., Subbiah, S., Wilkes, J.: Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In: Proceedings of the International Conference on Autonomic Computing (ICAC), pp. 69–82. USENIX (2013)
97. Napoli, C., Pappalardo, G., Tramontana, E.: A hybrid neuro–wavelet predictor for qoS control and stability. In: Baldoni, M., Baroglio, C., Boella, G., Micalizio, R. (eds.) AI*IA 2013. LNCS, vol. 8249, pp. 527–538. Springer, Heidelberg (2013)
98. Cloud-TM: Cloud-tm, d4.6: Final architecture (2013), `http://cloudtm.ist.utl.pt/cloudtm/final-deliverables/D4.6-ArchitectureReport.pdf`
99. Elnikety, S., Dropsho, S.G., Cecchet, E., Zwaenepoel, W.: Predicting replicated database scalability from standalone database profiling. In: Proceedings of EuroSys, pp. 303–316. ACM (2009)
100. Nicola, M., Jarke, M.: Performance modeling of distributed and replicated databases. IEEE Transactions on Knowledge and Data Engineering 12(4), 645–672 (2000)
101. di Sanzo, P., Antonacci, F., Ciciani, B., Palmieri, R., Pellegrini, A., Peluso, S., Quaglia, F., Rughetti, D., Vitali, R.: A framework for high performance simulation of transactional data grid platforms. In: Proceedings of the International Conference on Simulation Tools and Techniques (SimuTools), pp. 63–72. ACM (2013)
102. di Sanzo, P., Molfese, F., Rughetti, D., Ciciani, B.: Providing transaction class-based qos in in-memory data grids via machine learning. In: Proceedings of the Symposium on Network Cloud Computing and Applications (NCCA), pp. 46–53. IEEE (2014)
103. Didona, D., Quaglia, F., Romano, P., Torre, E.: Enhancing Performance Prediction Robustness by Combining Analytical Modeling and Machine Learning. In: Proceedings of the International Conference on Performance Engineering (ICPE). ACM (2015)