This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

Chasing the Optimum in Replicated In-memory Transactional Platforms via Protocol Adaptation

Maria Couceiro, Pedro Ruivo, Paolo Romano and Luís Rodrigues

Abstract—Replication plays an essential role for in-memory distributed transactional platforms, given that it represents the primary means to ensure data durability. Unfortunately, no single replication technique can ensure optimal performance across a wide range of workloads and system configurations. This paper tackles this problem by presenting MorphR, a framework that allows to automatically adapt the replication protocol of in-memory transactional platforms according to the current operational conditions. MorphR presents two key innovative aspects. On one hand, it allows to plug in, in a modular fashion, specialized algorithms to regulate the switching between arbitrary replication protocols. On the other hand, MorphR relies on state of the art machine learning techniques to autonomously determine the best replication in face of varying workloads. We integrated MorphR in an open-source in-memory NoSQL data grid, and evaluated it by means of an extensive experimental study. The results highlight that MorphR is accurate in identifying the best replication strategy in presence of complex realistic workloads, and does so with minimal overhead.

1 Introduction

With the advent of grid and cloud computing, in-memory distributed transactional platforms, such as NoSQL data grids [1] and Distributed Transactional Memory systems [2], [3], have gained an increased relevance. These platforms combine ease of programming and efficiency by providing transactional access to distributed shared state, and mechanisms aimed to elastically adjust the resource consumption (nodes, memory, processing) in face of changes in the demand.

In these platforms, replication plays an essential role, given that it represents the primary means to ensure data durability. The issue of transactional replication has been widely investigated in literature, targeting both classic databases [4], [5] and transactional memory systems [3]. As a result, a large number of replication protocols have been proposed, based on significantly different design principles, such as, single-master vs multi-master management of updates [6], [7], lock-based vs atomic broadcast-based serialization of transactions [2], [4], optimistic vs pessimistic conflict detection [3].

Unfortunately, as we clearly show in this paper, there is not a single replication strategy that outperforms all other strategies for a wide range of workloads and scales of the system. I.e., the best performance of the system can only be achieved by carefully selecting the appropriate replication protocol in function of the characteristics of the infrastructure (available resources, such as number of servers, CPU and memory capacity, network bandwidth, etc) and workload characteristics (read/write ratios, probability of conflicts, etc).

These facts raise two significant challenges. First, given that both resources and workloads are dynamic, the data grid platform must support the run-time change of replication protocols in order to achieve optimal efficiency. Second, the amount of parameters affecting the performance of replication protocols is so large, that the manual specification of adaptation policies is cumbersome (or even infeasible), motivating the need for fully autonomic, self-tuning solutions.

This paper addresses these issues by introducing MorphR, a framework supporting automatic adaptation of the replication protocols employed by in-memory transactional platforms. The contributions of this paper are the following:

- We present the results of an extensive performance evaluation study using an open source transactional data grid (Infinispan by Red Hat/JBoss), which we extended to support three different replication protocols, namely primary-backup [6], distributed locking based on two-phase commit [7], and total order broadcast based certification [4]. We consider workloads originated by both synthetic and complex standard benchmarks, and deployments over platforms of different scales. The results of our study highlight that none of these protocols can ensure the best performance for all possible configurations, providing a strong argument to pursue the design of abstractions and mechanisms supporting the online reconfiguration of replication protocols.
- We introduce a framework, named MorphR, which formalizes a set of interfaces with precisely defined semantics that need to be exposed (i.e. implemented) by an arbitrary replication protocol in order to support its online reconfiguration, i.e. switching to a different protocol. The proposed framework is designed to ensure both generality, by means of a protocol-agnostic generic reconfiguration protocol, and efficiency, whenever the cost of the transition between two specific replication protocols can be minimized by taking into account their intrinsic characteristics. We demonstrate the flexibility of the proposed reconfiguration framework by showing that it can seamlessly encapsulate the three replication protocols mentioned above, via both protocol-agnostic and specialized protocol switching techniques.
- We validate the MorphR framework, by integrating

- *M. Couceiro, P. Romano and L. Rodrigues are with INESC-ID and Instituto Superior Técnico, Universidade de Lisboa. P. Ruivo is with Red Hat Inc.*
- *This work has been partially supported by national funds through FCT (INESC- ID multiannual funding) through the PIDDAC Program funds, FCT projects PEst-OE/EEI/LA0021/2013, PPTDC/EEI-SCR/2776/2012.*

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

it in Infinispan, which allows to assess its practicality and efficiency in realistic transactional data grids. A noteworthy result highlighted by our experiments is that the MorphR-based version of Infinispan does not incur in perceivable performance overheads in absence of reconfigurations (which is expected to be the most frequent case), with respect to the non-reconfigurable version. We use this prototype to evaluate the latencies of generic and specialized reconfiguration techniques, demonstrating that the switching can be completed with a latency in the order of a few tens of milliseconds in a cluster of 10 nodes employing commodity-hardware.

• We show how to model the problem of determining the best replication protocol given the current operational conditions as a classification problem. Via an extensive experimental study, relying on three machine learning techniques and heterogeneous workloads and platform scales, we demonstrate that this learning problem can be solved with high accuracy.

The remainder of the paper is structured as follows. Section 2 reports the results of a performance evaluation study highlighting the relevance of the addressed problem. The system architecture is presented in Section 3 and its main components are presented in Sections 4 and 5. The experimental evaluation is reported in Section 6. Related work is analysed in Section 7. Section 8 concludes the paper.

2 Motivations

In the introduction above, we have stated that there is not a single replication strategy that outperforms all others. In this section, we provide the results of an experimental study backing this claim. Before presenting the experimental data, we provide detailed information on the experimental platform and on the benchmarks used in our study.

2.1 Experimental Platform

We used an open-source in-memory transactional data grid, namely Infinispan [8] by Red Hat/JBoss, as reference platform for this study. At the time of writing, Infinispan is the reference NoSQL platform and clustering technology for JBoss AS, a mainstream open source J2EE application server. From a programming API perspective, Infinispan exposes a key-value store interface enriched with transactional support. Infinispan maintains data entirely in memory, using a weakly consistent variant of a multi-version concurrency algorithm to regulate local concurrency. More in detail, the Infinispan prototype used in this work (namely, version 5.2.0), ensures two non-serializable consistency levels: repeatable read [9], and a variant that performs an additional test, called *write-skew check*, which aborts a transaction T whenever any of the keys T read and wrote is altered by any concurrent transaction [8]. In all the experiments reported in this paper, we select as consistency criterion the latter, stronger, consistency criterion.

Detection of remote conflicts, as well as data durability, are achieved by means of a Two Phase Commit [7] based replication protocol (2PC). To assess the performance of alternative transaction replication protocols, we developed two custom prototypes of Infinispan (ensuring the same consistency levels originally provided by Infinispan), in which we replaced the native replication protocol with two alternative protocols, i.e. Primary-Backup (PB) and a replication protocol based on Total Order Broadcast, which we refer to as TO. Note that due to the vastness of literature on transactional replication protocols, an exhaustive evaluation of all existing solutions is clearly infeasible. However, the three protocols that we consider, 2PC, PB, and TO, represent different well-known archetypal approaches, which have inspired the design of a plethora of different variants in literature. Hence, we argue that they capture the key tradeoffs in most existing solutions. However, the protocol-agnostic approach adopted by MorphR is flexible enough to cope with other replication protocols, including, e.g., partial replication and quorum protocols. Next, we briefly overview the three considered protocols:

2PC: Infinispan integrates a variant of the classic two phase commit based distributed locking protocol. In this scheme, transactions are executed locally in an optimistic fashion in every replica, avoiding any distributed coordination until the commit phase. At commit time, a variant of two phase commit is executed. During the first phase, updates are propagated to all replicas, but, unlike typical distributed locking schemes, locks are acquired only by a single node (called the “primary” node), whereas the remaining nodes simply acknowledge the reception of the transaction updates (without applying them). By acquiring locks on a single node, this protocol avoids distributed deadlocks, a main source of inefficiency of classic two phase commit based schemes. However, unlike the classic two phase commit protocol, the locks on the primary need to be maintained until all other nodes have acknowledged the processing of the commit. This protocol produces a large amount of network traffic, which typically leads to an increase of the commit latency (of update transactions), and suffers from a high lock duration, which can generate lock convoying at high contention levels.

PB: This is a single-master protocol allowing the processing of update transactions only on a single node, called the primary, whereas the remaining ones are used exclusively for processing read-only transactions. The primary regulates concurrency among local update transactions using a deadlock-free commit time locking strategy, and propagates synchronously its updates to the backup nodes. Read-only transactions can be processed in a non-blocking fashion on the backups, regulated by Infinispan’s native multiversion concurrency control algorithm. In this protocol, the primary is prone to become a bottleneck in write dominated workloads. On the other hand, its commit phase is simpler than in the other considered protocols (which follow a multi-master approach). This alleviates the load on the network and reduces the commit latency of update transactions. Further, by intrinsically limiting the number of concurrently active update transactions, it is less subject to trashing due to lock contention in high conflict scenarios.

TO: Similarly to 2PC, this protocol is a multi-master scheme that processes transactions without any synchronization during their execution phase. Unlike 2PC, however, the transaction serialization order is not determined by means of lock acquisition, but by relying on a Total Order Broadcast service (TOB) to establish a total order among committing transactions [10].

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

Upon their delivery by TOB, transactions are locally certified and either committed or aborted depending on the result of the certification. Being a lock-free algorithm, TO does not suffer from the lock convoying phenomena in high contention scenarios. However, its TOB-based commit phase imposes a larger communication overhead with respect to 2PC (and PB). This protocol has higher scalability potential than PB in write dominated workloads, but is also more prone to incur in high abort rates in conflict intensive scenarios.

2.2 Benchmarks

We consider three benchmarks representative of very heterogeneous domains, namely TPC-C [11], Radargun[1] and Geograph [12]. The former is a standard benchmark for OLTP systems, which portrays the activities of a wholesale supplier and generates mixes of read-only and update transactions with strongly skewed access patterns and heterogeneous durations. We have developed an implementation of TPC-C that was adapted to execute on a NoSQL key/value store, which include three different transaction profiles: Order Status, a read-only long running transaction; New Order, a computation intensive update transaction that generates moderate contention; and Payment, a short, conflict prone update transaction.

Radargun is a benchmarking framework designed by JBoss to test the performance of distributed, transactional key-value stores. The workloads generated by Radargun are simpler and less diverse than those of TPC-C, but they have the advantage of being very easily tunable, thus allowing to easily explore a wide range of possible workload settings.

Geograph is a benchmarking tool that allows for injecting complex and rich workloads representative of the most popular geo-social applications. Geograph has been designed to be flexible both in the heterogeneity and in the dynamics of the generated workloads. In particular, it provides 19 different geo-social services (i.e. actions) and 16 application specific user simulators — called agents. Agents can be combined in complex ways to generate dynamic workload profiles.

For TPC-C we consider three different workload scenarios, which are generated by configuring the following benchmark parameters: the number of warehouses, i.e. the size of the keyset that is concurrently accessed by transactions, which has a direct impact on the generated contention; the percentage of the mix of transaction profiles generated by the benchmark; and the number of active threads at each node, which allows to capture scenarios of machines with different CPU power (by changing the number of concurrent threads active on each node). This last parameter allows to simulate, for instance, changes of the computational capacity allocated to the virtual machines hosting the data platform in a Cloud computing environment. The detailed configuration of the parameters used to generate the three TPC-C workloads, which are referred to as TW1, TW2, and TW3, are reported in Table 1.

For Radargun we also consider three workloads, which we denote as RW1, RW2 and RW3. These three workloads are generated synthetically, and their characteristics can be controlled by tuning three parameters: the ratio of read-only vs update transactions; the number of read/write operations executed by (read-only/update) transactions; and the cardinality of the set of keys stored in the data grid — which are accessed uniformly at random by each read/write operation. The parameters settings used for these workloads is also depicted in Table 1.

1. https://github.com/radargun/radargun/wiki

Parameters Settings for the TPC-C Workloads

	# Warehouses	% Order Status	% Payment	% New Order	# Threads
TW1	10	20	70	10	1
TW2	1	20	30	50	8
TW3	1	30	70	0	1

Parameters Settings for the Radargun Workloads

	%Write Tx	# Reads RO Tx	# Reads Wrt Tx	# Writes (Wrt Tx)	# Keys	# Threads
RW1	50	2	1	1	5000	8
RW2	95	200	100	100	1000	8
RW3	40	50	25	25	1000	8

Parameters Settings for the Geograph Workloads

	Agent Composition	Location	Description	# Threads
GW1	128 Read-Post10UpdatePos90 Agents	128 different sites	Normal state of the application	12
GW2	196 Read-Post10UpdatePos90 Agents and 4 Blogger Agents	Rome	Big event	12
GW3	160 Read-Post10UpdatePos90 and 10 Blogger Agents	Rome	The event has finished	12

TABLE 1
Parameters Settings

As for Geograph, we consider a simulation of a sports fan geo-social application where fans can trace their trips to sport events and share their opinions about it. The workload simulates an application with a few hundreds of concurrent simulated agents who submit requests with 0 think time. This allows simulating actually a much larger population of users considering that typically each user updates its position/submits requests in geo-social networks with a frequency of tens of seconds/minutes. This is the workload that one would expect from a fairly popular application with large populations of registered users. Table 1 reports the parameters settings used for the considered workloads. These experiments relied on two different types of agents: *bloggers*, which make small text posts (Twitter-style) without commenting, liking and searching for posts; and *ReadPost10UpdatePos90*, a type of agent that tracks its current position while reading posts in the vicinity (in this case, it reads posts 10% of the time and updates the position 90% of the time). Regarding the workloads, GW1 simulates a steady scenario in which users are spread across the world in 128 different locations. GW2 simulates the start of a major sport event in Rome, during which many users share their position, read posts about the surroundings, and comment on the event live. In GW3, the ceremony is over and the number of agents who share their position starts diminishing while more agents start blogging about the event.

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

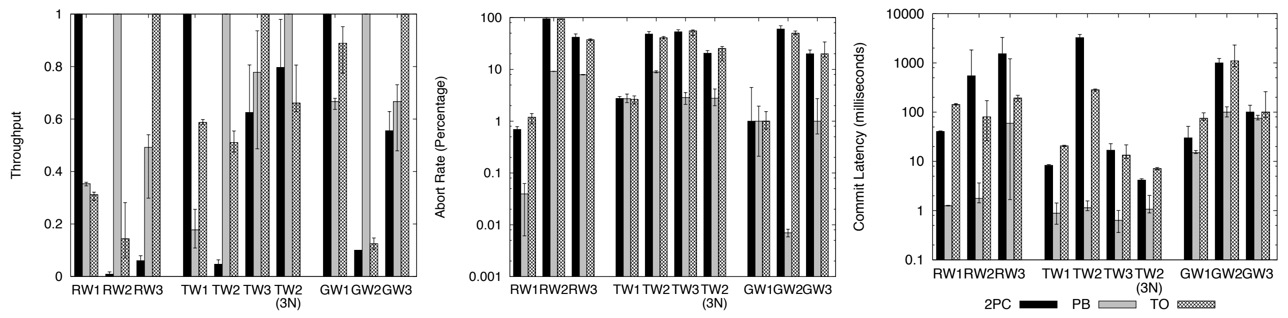


Fig. 1. Performance of 2PC, PB and TO protocols.

2.3 Analysis of the results

We now report and discuss experimental data that illustrates the performance of 2PC, PB, and TO protocols under different workloads. All the experiments reported in this paper have been obtained using a commodity cluster of 10 nodes. The machines are connected via a Gigabit switch and each one has Ubuntu 10.04 installed, 8 cores (2 Intel(R) Xeon(R) CPU E5506 @ 2.13GHz) and 32GB of RAM memory. We performed experiments with different cluster sizes. However, as the corresponding data sets show similar trends, for space constraints, for most workloads we only report results using 10 nodes; the only exceptions are workload TW2, for which we also depict results in a 3 nodes cluster (denoted as TW2(3N)), and Geograph, for which we used a cluster of 5 nodes.

The first plot in Figure 1 reports the throughput achieved by each protocol normalized to the throughput of the best performing protocol (in the same scenario). The second and third plots report values on the transaction abort rate and commit latency (both using logscale for the y axis).

The results clearly show that none of the protocols can achieve optimal performance in all the considered workload configurations. Furthermore, the relative differences among the performance of the protocols can be remarkably high: the average normalized throughput of the worst performing protocol across all workloads is around 20% (i.e. one fifth) of the the best performing protocol for each workload; also, the average throughputs across all workloads of the PB, TO, and 2PC are approximately, respectively, 30%, 40% and 50% lower than that of the optimal protocol. Furthermore, by contrasting the results obtained with TW2 using different scales of the platform, it can be observed that, even for a given fixed workload, the best performing replication protocol can be a function of the number of nodes currently in the system. These figures provide a direct measure of the potential inefficiency of a statically configured system.

The reasons underlying the shifts in the relative performance of the replication protocols can be quite intricate, as the performance of the considered protocols is affected by a number of inter-dependent factors affecting the degree of contention on both logical (i.e. data) and physical (mostly network and CPU) resources. Therefore, it may be extremely hard to manually define policies that control the adaptation. Next, we provide some insights on these factors.

In the workloads RW1, TW1 and GW1, 2PC has a good leverage over the remaining protocols. This is explainable considering that, in these low conflict configurations, 2PC does not suffer of lock convoying. This allows it to leverage on its ability to process update transactions at all nodes (unlike PB), while enjoying lower commit latencies w.r.t. TO.

The workloads RW2, TW2 and GW2, conversely, are favourable to PB in both these scenarios. In fact, the high degree of contention causes the thrashing of the two multi-master protocols (2PC and TO), which suffer from an extremely high transaction abort rate.

In the scenarios favourable to TO, namely RW3, TW3 and GW3, contention is sufficiently high to cause lock convoying on 2PC, which results in a higher commit latency. Furthermore, the workloads contain a sufficient percentage of update transactions to saturate the primary node with PB.

3 Architectural Overview

The architecture of MorphR is depicted in Figure 2. The system is composed by two macro components, a *Reconfigurable Replicated In-Memory Data Store* (RRITS), and a *Replication Protocol Selector Oracle* (RPSO).

The RRITS externalizes user-level APIs for transactional data manipulation (such as those provided by a key/value store, as in our current prototype, or an STM platform), as well as APIs, used in MorphR by the RPSO, that allow its remote monitoring and control (to trigger adaptation). Internally, the RRITS supports multi-layer reconfiguration techniques, which are encapsulated by abstract interfaces allowing to plug in, in a modular fashion, arbitrary protocols for replica coherency and concurrency control. The Group Communication System (GCS) coordinates the communication between all nodes in the system and detects any faults that may occur. A detailed description of this building block is provided in Section 4.

The RPSO is an abstraction that allows encapsulating different performance forecasting methodologies. The Remote Monitoring and Control component relies on JMX to bridge interactions between RPSO and the RRITS for collecting statistics to feed the oracle and triggering a protocol switch. The oracle implementation may be centralized or distributed. In a centralized implementation, the RPSO is a single process that runs in one of the replicas or in a separate machine. In the distributed implementation, each replica has its own local instance of the oracle that coordinates with other instances to reach a common decision. In this work, we chose to implement

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

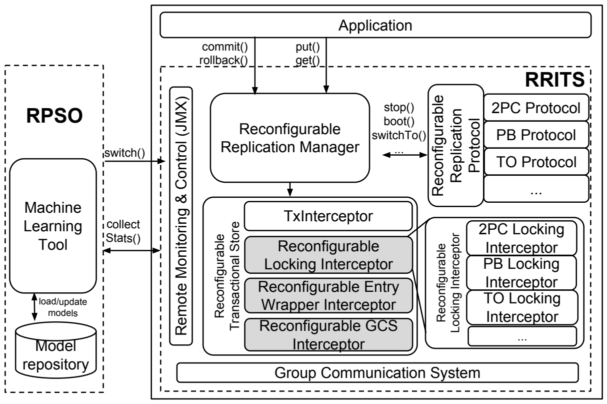

Application
commit()
rollback()
put()
get()
RRITS
RPSO
switch()
Machine Learning Tool
collect Stats()
load/update models
Model repository
Remote Monitoring & Control (JMX)
Reconfigurable Replication Manager
stop()
boot()
switchTo()
...
Reconfigurable Replication Protocol
2PC Protocol
PB Protocol
TO Protocol
...
Reconfigurable Transactional Store
TxInterceptor
Reconfigurable Locking Interceptor
Reconfigurable Entry Wrapper Interceptor
Reconfigurable GCS Interceptor
Reconfigurable Locking Interceptor
2PC Locking Interceptor
PB Locking Interceptor
TO Locking Interceptor
...
Group Communication System

Fig. 2. Architectural overview.

the centralized version, which will be discussed in more detail in Section 5.

4 In-Memory Data Storage

The RRITS is composed by two main sub-components, the Reconfigurable Replication Manager and the Reconfigurable Transactional Store, which are described next.

4.1 Reconfigurable Replication Manager

The Reconfigurable Replication Manager (RRM) is the component in charge of orchestrating the reconfiguration of the replication protocol, namely the transition from a state of the system in which transactions are processed using a replication protocol A, to a state in which they are processed using a protocol B. The design of RRM was guided by the goal of achieving both generality and efficiency.

An important observation is that in order to maximize efficiency, it is often necessary to take a *white box* approach: by exploiting knowledge on the internals of the involved protocols, it is usually possible to define specialized (i.e. highly efficient) reconfiguration schemes. On the other hand, designing specialized reconfiguration algorithms for all possible pairs of protocols leads to an undesirable growth of complexity, which can hamper the platform's extensibility.

MorphR addresses this tradeoff by introducing a generic, protocol-agnostic reconfiguration protocol that guarantees the correct switching between two arbitrary replication protocols, as long as these adhere to a very simple interface (denoted as *ReconfigurableReplicationProtocol* in Figure 2). This interface allows MorphR to properly control their execution (stop and boot them). In order to achieve full generality, i.e. to be able to ensure consistency in presence of transitions between any two replication protocols, MorphR's generic reconfiguration protocol is based on a conservative "stop and go" approach, which enforces the termination of all transactions in the old protocol, putting the system in a quiescent state, before starting executing the new protocol.

MorphR requires that all pluggable protocols implement the methods needed by this stop and go strategy (described below), benefiting from its extensibility and guaranteeing the generality of the approach. On the other hand, in order to

```
stop(boolean eager) {
    block generation of new local transactions;
    if eager then
        abort any local executing transaction;
    else
        wait for completion of all local executing transactions;
    end
    broadcast (DONE);
    wait received DONE from all processes;
    wait for completion of all remote transactions;
}
```

Algorithm 1: *stop*() method of the 2PC protocol.

maximize efficiency, for each pair of replication protocols (A,B), MorphR allows for registering an additional *protocol switcher* algorithm, which interacts with the RRM via a well defined interface. The RRM also uses such specialized reconfiguration protocols, whenever available, and otherwise resorts to using the protocol-agnostic reconfiguration scheme.

Figure 3 depicts the state machine of the reconfiguration strategies supported by MorphR. Initially the system is in the STEADY state, running a single protocol A. When a transition to another protocol B is requested, two paths are possible. The default path (associated with the generic "stop and go" approach) first puts the system in the QUIESCENT state and then starts protocol B, which will put the system back to the STEADY state. The fast path consists of invoking the fast switching protocol. This protocol will place the system in a temporary TRANSITION state, where both protocol A and protocol B will coexist. When the switch terminates, only protocol B will be executed and the system will be again in the STEADY state. For instance, the system may first switch A→B using the fast switch, and then perform the transition B→C using a stop and go switch (because it may be very hard or impossible to implement a fast switch transition B→C).

We will now discuss, in turn, each of the two protocol reconfiguration strategies supported by MorphR.

"Stop and Go" reconfiguration: The methods defined in the *ReconfigurableReplicationProtocol* interface can be grouped in two categories: i) a set of methods that allow the RRM to catch and propagate the transactional data manipulation calls issued by the application (e.g. begin, commit, abort, read and write operations), and ii) two methods, namely *boot()* and *stop()*, that every pluggable protocol must implement:

- *boot()*: This method is invoked to start the execution of a protocol from a QUIESCENT state, i.e., no transactions from any other protocol are active in the system, and implements any special initialization conditions required by the protocol.
- *stop*(*boolean* eager): This method is used to stop the execution of a protocol and putting the system in a QUIESCENT state. The protocol dependent implementation of this method must guarantee that, when it returns, there are no transactions active in the system executing with that protocol. The *eager* parameter is a boolean that allows to specify if on-going transactions should be aborted immediately, or if the system should allow for on-going transactions to terminate before entering the QUIESCENT state.

Algorithm 1 exemplifies the implementation of this interface, for the case of the 2PC. First, all new local transactions

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

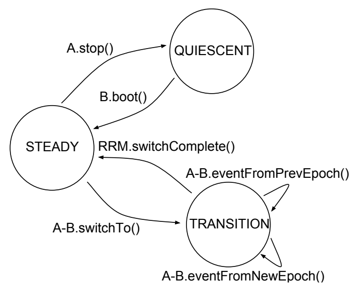


Fig. 3. MORPHR reconfiguration finite state machine.

are blocked. When local transactions finish executing, a DONE message is broadcast announcing that no more transactions will be issued by this replica in the current protocol. Before returning, the *stop* method waits for this message from all the other replicas in the system and for the completion of any remote transactions executing in the that replica.

Fast switching reconfiguration: The default "stop and go" strategy ensures that, at any moment in time, no two transactions originated by different protocols can be concurrently active in the system. Non-blocking reconfiguration schemes avoid this limitation by allowing overlapping the execution of different protocols during the reconfiguration. In order to establish an order on the reconfigurations, the RRM (i.e. each of the RRM instances maintained by the nodes in the system) relies on the notion of epochs. Each fast switching reconfiguration triggers a new epoch and all transaction events (namely, prepare, commit, and abort events) are tagged with the epoch in which they were generated.

To support fast switching between a given pair of protocols (oldProtocol, newProtocol), the MORPHR framework requires that the programmer implements the following set of methods:

- *switchTo()*: This method is invoked to initiate fast switching. This triggers the increase of the local epoch counter on the replica, and alters the state of the replica to TRANSITION.
- *eventFromPrevEpoch*(event): This method processes an event of a transaction that was initiated in an epoch previous to the one currently active in this replica.
- *eventFromNewEpoch*(event): This method processes an event from a transaction that was initiated by a replica that is either in the TRANSITION state of the new epoch, or that has already completed the switch to new epoch and entered the STEADY state.

As also depicted by the state machine in Figure 3, the methods *eventFromPrevEpoch* and *eventFromNewEpoch* are only executed by a replica that has entered the TRANSITION state. Hence, whenever a replica receives either one of these two events while it is still in the STEADY state of protocol A, it simply buffers them, delaying their processing till the *switchTo()* method is invoked[2].

Further, the RRM exposes a callback interface, via the *switchComplete*() method, which allows the protocol switcher

```
1  2PC-PB.switchTo() {
2      broadcast (LOCALDONE);
3  }
4  2PC-PB.eventFromPrevEpoch(event) {
5      processEvent(event, tx);
6  }
7  2PC-PB.eventFromNewEpoch(event) {
8      processEvent(event, tx);
9  }
10 upon received LOCALDONE from all nodes {
11     wait for completion of prepared remote 2PC txs;
12     // guarantee reconfiguration completion globally
13     broadcast (REMOTEDONE);
14     wait received REMOTEDONE from all nodes;
15     switchComplete();
16 }
```

Algorithm 2: Fast Switching from 2PC to PB.

to notify the end of the transition phase to the RRM, and which causes it to transit to the STEADY state. As for the *stop()* method, a *protocol switcher* implementation must guarantee that when the *switchComplete*() method is invoked, every transaction active in the system is executing according to the final protocol. In the following paragraphs we provide two examples of fast switching algorithms, for scenarios involving protocols whose concurrent coexistence raises different types of issues, namely 2PC→PB and 2PC→TO.

Fast switch from 2PC to PB: Both PB and 2PC are lock based protocols. Further, in both protocols, locks are acquired only on a designated node, called primary (both in 2PC and PB). Hence, provided that these two nodes coincide (which is the case, for instance, in our Infinispan prototype), these two specific protocols can simply coexist, and keep processing their incoming events normally. Algorithm 2 shows the pseudo-code of the fast switching for this case. As the two protocols can seamlessly execute in parallel, in order to comply with the specification of the *fast switching* interface, it is only necessary to guarantee that when the *switchComplete* callback is invoked, no transaction in the system is still executing using 2PC. To this end, when the switching is started, a LOCALDONE message is broadcast and the protocol moves to a TRANSITION state, causing the activation of a new epoch. In the TRANSITION state, locally generated transactions will be already processed using PB. When the LOCALDONE message is received from node s by some node n, it derives from the FIFO property of channels that n will not receive additional prepare messages from s. By collecting LOCALDONE message from each and every node in the system, each node n can attest the local termination of the previous epoch, at which point it broadcasts a REMOTEDONE message (line 13). The absence of transactions currently executing with 2PC across the entire system can then be ensured by collecting the latter messages from all nodes (see lines 14-15).

Fast switch from 2PC to TO: 2PC and TO protocols are radically different, as they use different concurrency control schemes (lock-based vs lock-free) and communication primitives (plain vs totally ordered broadcast) that require different data-structures/algorithms at the transactional data store level.

2. As an optimization, in this case our prototype actually avoids buffering *eventFromPrevEpoch* events: this is safe because it means that the transaction's event has been generated in the same epoch as the one in which the local node is currently executing.

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

```
2PC-TO.switchTo() {
    block generation of new local transactions;
    wait for local transactions in prepared state;
    broadcast (LocalDone);
}
2PC-TO.eventFromPrevEpoch(event) {
    if event is of type Prepare then
        rollback(tx);
    end
    processEvent(event, tx);
}
2PC-TO.eventFromNewEpoch(event) {
    if tx conflicts with some tx' that uses 2PC then
        wait for tx' to commit or abort;
    end
    processEvent(event, tx);
}
upon received LocalDone from all nodes {
    // guarantee reconfiguration completion globally
    broadcast (RemoteDone);
    wait received RemoteDone from all nodes;
    switchComplete();
}
```
Algorithm 3: Fast switching from 2PC to TO.

So, it is impossible for a replica to start processing transactions with TO if any locally generated 2PC transaction is still active. To this end, the fast switch implementation from 2PC to TO, in Algorithm 3, returns from the *switchTo* method (entering the new, TO-regulated epoch) only after it has committed (or aborted) all its local transactions from the current epoch. During the TRANSITION state, a node replies negatively to any incoming prepare message for a remote 2PC transaction, thus avoiding incompatibility issues with the currently executing TO protocol. Transactions from the new TO epoch, on the other hand, can be validated (and accordingly committed or aborted). However, if they conflict with any previously prepared but not yet committed 2PC transaction, the commit of the TO transaction must be postponed until the outcome of previous 2PC transactions is known. Otherwise, it can be processed immediately according to the conventional TO protocol. Also in this case a second global synchronization phase is required to ensure the semantics of the *switchComplete*.

Coping with failures: MorphR ensures fault tolerance as long as its three main elements are fault tolerant, namely the replication protocols, RPSO, and RRITS:

• Replication protocols must (and typically do) implement their own recovery schemes; for instance, in 2PC, if a node crashes, the protocol must be able to deal appropriately with this scenario, in order to ensure that locks are eventually released. Since the current implementation of MorphR relies on a group membership service, nodes leverage on this building block to detect and recover from failures.

• As for the RPSO, standard replication techniques, such as primary-backup or quorum-based approaches, can be used to ensure the oracle's fault-tolerance.

• Concerning the RRITS, the algorithms that govern both the ”stop-and-go” and the ”fast-switching” schemes were presented assuming no faults for simplicity. They contain various wait conditions that require gathering messages from all replicas; in order to avoid blocking arbitrarily on these wait conditions, a simple approach is to rely on the group membership service and wait for messages from all the nodes currently in the group.

4.2 Reconfigurable Transactional Store

MorphR assumes that, when a new replication protocol is activated, the *boot()* method performs all the setup required for the correct execution of that protocol. In some cases, this may involve performing some amount of reconfiguration of the underlying data store, given that the replication protocol and the concurrency control algorithms are often tightly coupled. Naturally, this setup is highly dependent of the concrete data store implementation in use.

When implementing MorphR on Infinispan, our approach to the protocol setup problem has been to extend the original Infinispan architecture in a principled way with the aim to minimize intrusiveness. To this end, we systematically encapsulated the modules of Infinispan that required reconfiguration using software *wrappers*. The wrappers intercept calls to the encapsulated module, and re-route them to the implementation associated with the current replication protocol configuration.

The architectural diagram in Figure 2 illustrates how this principle was applied to one of the key elements of Infinispan, namely the interceptor chain that is responsible for i) capturing commands issued by the user and by the replication protocols, and ii) redirecting them towards the modules managing specific subsystems of the data store (such as the locking system, the data container, or the group communication system). The interceptors whose behaviours had to be replaced due to an adaptation of the replication protocol, shown in gray in Figure 2, were replaced with generic reconfigurable interceptors, for which each replication protocol can provide its own specialized implementation. This allows to flexibly customize the behaviour of the data container depending on the replication protocol currently in use.

5 Selector Oracle

The Replication Protocol Selector Oracle component is a convenient form of encapsulating different performance forecasting techniques. In fact, different approaches, including analytical models [13] and machine learning (ML) techniques [14], [15], might be adopted to identify the replication protocol on the basis of the current operating conditions. In MorphR we have opted for using ML-based forecasting techniques, as they can cope with arbitrary replication protocols, maximizing the generality and extensibility of the proposed approach, thanks to their black-box nature.

The selection of the best replication protocol lends itself naturally to be cast as a *classification* problem [14], in which one is provided with a set of input variables (also called *features*) describing the current state of the system and is required to determine, as output, a value from a discrete domain (i.e., the best performing protocol among a finite set in our case). We integrated MorphR with three distinct ML-based classification techniques: the C5.0 [16] decision tree algorithm, Neural Networks (NN) [17], and Support Vector

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

Machines (SVM) [18]. C5.0 builds a decision-tree classification model during an off-line training phase in which a greedy heuristic is used to partition, at each level of the tree, the training dataset according to the input feature that maximizes information gain [16]. The output model is a decision-tree that closely classifies the training cases according to a compact (human-readable) rule-set, which can then be used to classify (i.e., decide the best performing replication strategy for) future scenarios. Regarding NN and SVM, we used the implementations available in Weka [19], a popular open-source framework that provides a common interface to a large number of ML algorithms. The NN algorithm implemented in the Weka framework trains a multi-layered network using the classic back-propagation algorithm [20] to classify instances. We used the default configuration in Weka, which generates a number of hidden layers equal to half the number of input features. Concerning the Support Vector Machine technique, we also rely on the default configuration of the Weka's SMO package, which implements John Platt's sequential minimal optimization algorithm for training a support vector classifier [21]. We shall discuss the methodology adopted in MORPHR to build ML-based performance models shortly, and focus for the moment on discussing how these models are used at runtime.

In our current reference architecture, the RPSO is a centralized logical component, which is physically deployed on one of the replicas in the system. Although the system is designed to transparently support the placement of the RPSO on a dedicated machine, the overhead imposed to query the ML model is so limited (on the order of the microseconds), and the query frequency is so low (on the order of the minutes or of the tens of seconds), that the RPSO can be collocated on any node of the data platform without causing perceivable performance interferences.

The RPSO periodically queries each node in the system, gathering information on several metrics describing different characteristics of the current workload in terms of both contention on logical (data) and physical resources. This information is transformed into a set of input features used to query the machine learner about the most appropriate configuration. If the current protocol configuration matches the predicted one, no action is taken; otherwise a new configuration is triggered.

This approach results in an obvious tradeoff: the more often the RPSO queries the ML, the faster it reacts to changes in the workloads. However, it also increases the risk to trigger unnecessary configuration changes upon the occurrence of momentary spikes that do not reflect a real sustained change in the workload. In our current prototype we use a simple approach based on a moving average over a window time of 30 seconds, which has proven successful with all the workloads we experimented with. As with any other autonomic system, in MORPHR there is also a tradeoff between how fast one reacts to changes and the stability of the resulting system[3]. In the current prototype, we simple use a fixed "quarantine" period after each reconfiguration, to ensure that the results of the previous adaptation stabilise before new adaptations are evaluated. Of course, the system may be made more robust by introducing techniques to filter out outliers [22], detect statistically relevant shifts of system's metrics [23], or predict future workload trends [24]. These techniques are orthogonal to the contributions of this paper, and fall outside of its scope.

3. Stability refers to the unlikelihood that a transient workload oscillation induces a spurious protocol switch.

Construction of the ML model. The accuracy achievable by any ML technique is well known to be strictly dependent on the selection of appropriate input features [14]. These should be, on one hand, sufficiently rich to allow the ML algorithm to infer rules capable of closely relating fluctuations of the input variables with shifts of the target variable. On the other hand, considering an excessively high number of features leads to an exponential growth of the training phase duration and to an increase of the risk of inferring erroneous/non-general relationships (a phenomenon called over-fitting [14]).

After conducting an extensive set of preliminary experiments, we decided to let MORPHR gather a base set of 14 metrics, namely: percentage of write transactions, number of read and write operations per read-only and write transactions and their local and total execution time, abort rate, throughput, lock waiting time and hold time, duration of the commit phase, average CPU and memory utilization.

As we will show in Section 6, this set of input features proved sufficient to achieve high prediction accuracy, at least for the set of replication protocols considered in this paper. Nevertheless, to ensure the generality of the proposed approach, we allow protocol developers to enrich the set of input features for the ML by specifying, using an XML descriptor, whether the RPSO should track any additional metric that the protocol exposes via a standard JMX interface.

A key challenge to address in order to build accurate ML-based predictors of the performance of multiple replication protocols is that several of the metrics measurable at run-time can be strongly affected by the replication protocol currently in use. Let us consider the example of the transaction abort rate: in workloads characterized by high data contention, the 2PC abort rate is typically significantly higher than when using the PB protocol for the same workload, due to the presence of a higher number of concurrent (and distributed) writers. In other words, the input features associated with the same workload can be significantly different when observed from different replication protocols. Hence, unless additional information is provided that allows the ML to contextualize the information encoded in the input features, one risks feeding the ML with contradictory inputs that can end up confusing the ML inference algorithm and ultimately hinder its accuracy.

In order to address this issue, we consider three alternative strategies for building ML models: i) a simple baseline scheme, which does not provide any information to the ML concerning the currently used replication protocol; ii) an approach in which we extend the set of input features with the protocol used while gathering the features; iii) a solution based on the idea of using (training and querying) a distinct model for each different replication protocol. The second approach is based on the intuition that, by providing information concerning the "context" (i.e., the replication protocol) in which the input features are gathered, the ML algorithm may use

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

this information to disambiguate otherwise misleading cases. The third approach aims at preventing the problem *a priori*, by avoiding the usage of information gathered using different protocols in the same model. An evaluation of these alternative strategies can be found in Section 6.1.

Finally, the last step of the model building phase consists in the execution of an automated feature selection algorithm, which is aimed at minimizing the risk of overfitting and maximizing the generality of the model by discarding features that are either too closely correlated among each other (and hence redundant), or too loosely correlated with the output variable (and hence useless). We rely on the Forward Selection [25] technique, a greedy heuristic that progressively extends the set of selected features till the accuracy it achieves when using ten-fold cross-validation on the training set is maximized.

6 Evaluation

This section presents the results of an experimental study aimed at assessing three main aspects: the accuracy of the ML-based selection of the best-fitting replication protocol (Section 6.1); the efficiency of the alternative protocol switching strategies discussed in Section 4 (Section 6.2); the overheads introduced by the online monitoring and re-configuration supports employed by MorphR to achieve self-tuning (Section 6.3).

6.1 Accuracy of the RPSO

In order to assess the accuracy of the RPSO with the three different ML approaches, we generated approximately 75 workloads for both TPC-C and Radargun (results for Geograph are omitted for space constraints and because they show similar trends), varying uniformly the parameters that control the composition of transaction mixes and their duration. In particular, for each of the 3 TPC-C workloads described in Section 2, we explored 25 different configurations, varying the percentages of Order Status, Payment and New Order Transactions. Analogously, starting from each of the three Radargun workloads reported in Table 1, we explored 27 different variations of the parameters that control the percentage of write transactions, and the number of read/write operations both in read-only and update transactions. This workload generation strategy allowed us to obtain a balanced data set containing approximately the same number of workloads for which each of the three considered protocols results to be the optimal choice.

We ran each of the above workloads with the three considered protocols, yielding a data set of approximately 1350 cases that serves as the basis for this study. Figure 4 provides an interesting perspective on our data set, reporting the normalized performance (i.e., committed transactions per second) of the 2nd and 3rd best choice with respect to the optimal protocol (i.e., the protocol that had the best performance) for each of the considered workloads. The plot highlights that, when considering the Radargun workloads, the selection of the correct protocol configuration has a striking effect on system's throughput: in 50% of the workloads, the selection of the 2nd best performing protocol is at least twice slower than the optimal protocol; further, the performance decreases

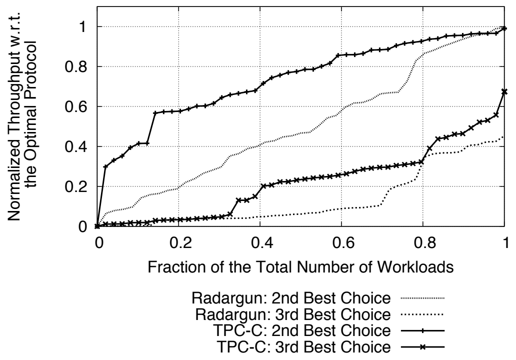


Fig. 4. Cumulative distribution of the normalized throughput vs the optimal protocol.

ML	Model Type	Radargun	TPCC
C5.0	w/o Prot.	9.90%	17.40%
	With Prot.	3.30%	13.30%
	Three Models	3.30%	10.00%
SVM	w/o Prot.	14.4%	15.06%
	With Prot.	14.4%	15.52%
	Three Models	12.66%	12.32%
NN	w/o Prot.	5.43%	11.87%
	With Prot.	3.7%	12.32%
	Three Models	2.88%	9.13%

TABLE 2
Percentage of misclassification.

by a factor up to 10x in 50% of the workloads in case the worst performing protocol were to be erroneously selected by the RPSO. On the other hand, the TPC-C workload shows less dramatic, albeit still significant, differences in the relative performances of the protocols. Being the performance of the three protocols relatively closer with TPC-C than with Radargun, the classification problem at hand is indeed harder in the TPC-C scenario, at least provided that one evaluates the accuracy of the ML-based classifier exclusively in terms of misclassification rate. On the other hand, in practical settings, the actual relevance of a misclassification is clearly dependent on the actual throughput loss due to the sub-optimal protocol selection. In this sense, the Radargun's workloads are significantly more challenging than TPC-C's ones. Hence, whenever possible, we will evaluate the quality of our ML-based classifiers using both metrics, i.e. misclassification rate and throughput loss vs optimal protocol.

The first goal of our evaluation is to assess the accuracy achievable by using the three alternative model building strategies described in Section 5, namely i) a baseline that adopts a single model built using no information on the protocol in execution when collecting the input features, ii) an approach in which we include the protocol currently in use among the input features, and iii) a solution using a distinct model per each protocol.

Table 2 shows the percentage of misclassification for each of the above approaches and for each of the three considered ML algorithms. These results were obtained by averaging the results of ten models (for each ML algorithm), each built using

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

ML	Train	Radargun		TPCC	
		Miscl.	**Thr. Loss**	**Miscl.**	**Thr. Loss**
C5.0	**90%**	12%	1%	18%	1%
	70%	15%	2%	22%	4%
	40%	14 %	4%	25%	9%
SVM	**90%**	8%	0.3%	18%	1%
	70%	17%	7%	14%	2%
	40%	13%	2%	17%	1%
NN	**90%**	0%	0%	27%	5%
	70%	5%	1%	9%	2%
	40%	8%	0.5%	12%	2%
2nd prot.		100%	14%	100%	23%
3rd prot.		100%	51%	100%	75%
Random Choice		33%	45%	33%	35%

TABLE 3
Accuracy of the considered ML algorithms.

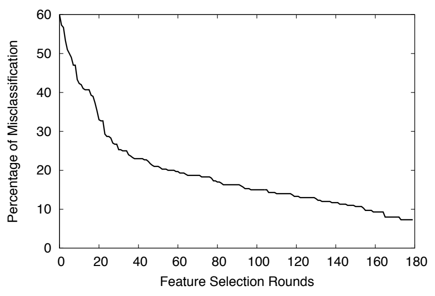


Fig. 5. Misclassification vs feature selection.

ten-fold cross validation. The results show that the misclassification rate can be significantly lowered by incorporating in the model information on the protocol in use when characterizing the current workload and system's state. In particular, using distinct models for each protocol, as expected, we minimize the chances that the ML is misled by the simultaneous presence of training cases exhibiting similar values for the same feature but associated with different optimal protocols (due to being measured when running different protocols), or, vice versa, of training cases associated with the same optimal protocol but exhibiting different values for the same feature (again, because they were observed using different protocols). At the light of this result, in MorphR, as well as in the remainder of this section, we opted for using a distinct model for each protocol.

The data in Table 3 allows us also to compare the three considered ML methodologies from the two-fold perspective of absolute and relative accuracy in the selection of the replication protocol, by reporting the misclassification rate (i.e., the percentage of wrongly classified workloads) and the relative loss in throughput with respect to the optimal protocol. The table presents information concerning models based on training sets of different sizes, as well as for the 2nd and 3rd best performing protocol for each scenario, and for a trivial uniformly random selection strategy.

The data highlights that NN achieves globally the highest accuracy, both in absolute and relative terms, with C5.0 and SVM closely following. Interestingly, when looking at the average throughput loss due to the choice performed by the predictive models, one can notice that all the considered models achieve efficiency levels very close to the optimal one. The fact that the misclassification rate is typically significantly larger than the average throughput loss w.r.t. the optimum means that, in the cases in which the predictive models misclassify a workload, the selected protocol delivers a performance relatively close to the optimal choice. These are scenarios in which, on one hand, it is naturally harder to predict which protocol delivers the best performance, given that the two best protocols achieve very similar performance. On the other hand, precisely because of this, in these scenarios the relative gain achievable by using a "perfect" model (one that is always correct) w.r.t. either of the considered ML algorithms is typically very modest and always less than 10%. Finally, the last three rows of the table confirm that the selection of the correct replication protocol can have a strong impact on system's performance, at least for the set of workloads considered in this study. This perspective allows for better appreciating the actual effectiveness of the considered ML-based predictors. Overall, the fact that all the considered ML algorithms achieve a quite good, and relatively close, accuracy suggests that, at least considering the dataset used in this study, the problem of identifying the best performing protocol for a given workload lends itself nicely to solutions based on statistical, black-box approaches.

In order to assess the time each ML technique requires to build a model, we used the entire data set for the Radargun benchmark and ran the previously mentioned Forward Selection technique to obtain the most accurate model. C5.0 was the fastest algorithm, taking 22 seconds, while SVM and NN were 14x and 17x slower, respectively, to build the best model. When considering these figures, however, one should take into account that C5.0 is a commercial quality product implemented in C, whereas the NN and SVM implementations used in this study are coded in Java and are part of a research-oriented framework (Weka) designed to ensure extensibility and flexibility rather than maximizing performance.

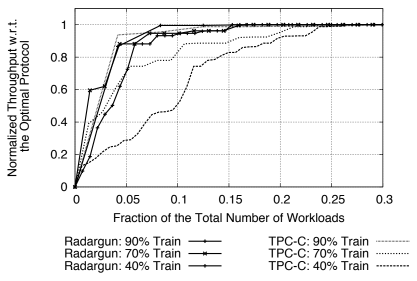
In Figure 5 we show data highlighting the relevance of choosing the most appropriate combination of features when building ML based models. To this end the plot reports the misclassification rate achieved when using models (w/o protocol information, using C5.0 and the TPC-C benchmark) based on the several combinations of features tested during the feature selection process. The plot clearly shows the impact of the correct selection of the set of features during the model construction phase, which is in practice the most time consuming one as it typically requires generating and comparing tens of different models.

Figure 6 evaluates the accuracy of the RPSO from a different perspective, reporting the cumulative distribution of the throughput achievable by the RPSO's predictions for each workload, normalized to the throughput of the optimal protocol for that workload. In order to assess the extrapolation power of the classifiers built using the proposed methodology we progressively reduce the training set from 90% to 40% of the entire data set, and use the remaining cases as test sets.

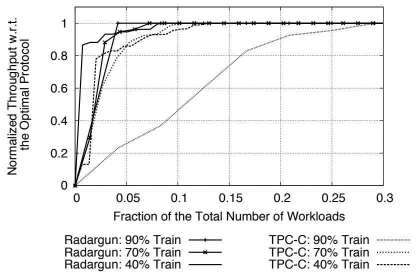
Both benchmarks show the same trend for all ML approaches. As the training set becomes larger, the percentage

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

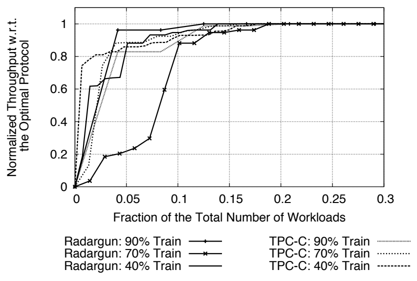
This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460



(a) C5.0



(b) NN



(c) SVM

Fig. 6. Cumulative distribution of the normalized throughput vs the optimal protocol.

of cases with sub-optimal throughput decreases. Furthermore, in these cases, the loss of throughput in absolute value, when compared to the optimal choice, also decreases. In fact, even for models built using the smallest training set, and considering the most challenging benchmark (namely TPC-C), the performance penalty with respect to the optimal configuration is lower than 10% in around 85% of the workloads. On the other hand, for models built using the largest training set, the throughput penalty is lower than 10% in about 90% of the cases. Again, the plots confirm that the accuracy achieved by the three ML approaches is, globally, quite similar.

Overall, the data highlights the remarkable accuracy achievable by the proposed ML-based forecasting methodology, providing an experimental evidence of its practical viability even with complex benchmarks such as TPC-C.

6.2 Fast switch *vs* Default Switch

We now analyse the performance of specialized fast switching algorithms, contrasting it with that achievable by the generic, but less efficient, stop and go switching approach. For this purpose we built a simple synthetic benchmark designed to generate transactions with tunable duration, varying from 15 milliseconds to 20 seconds. Furthermore, we have experimented with different fast switching algorithms and both with the eager and the lazy versions of the stop and go algorithm (recall that with the eager version ongoing transactions are simply aborted, whereas with the lazy version we wait for pending transactions to terminate before switching).

We start by considering the fast switching specialized that commutes from 2PC to PB (Alg. 2). Figure 7(a) shows the average blocking time, i.e., the period during which new transactions are not accepted in the system due to the switch (the shorter this period the better); the y axis is presented in logscale. Figure 7(b) shows the abort rate during the switching process. The figures show values for the fast-switching algorithm, and for both the lazy and eager version of stop-and-go.

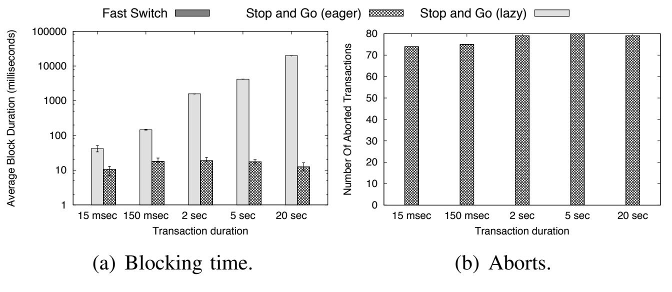
The fast switching algorithm has no blocking phase and for the scenarios where the duration of transactions is larger, this can be a huge advantage when compared with the lazy stop and go approach, which in the eager version has the lowest blocking time of 10ms. As expected, the fast switching algorithm is independent of the transaction duration, as it is not required to abort or to wait for the termination of transactions started with 2PC before accepting transactions with PB. On the other hand, the lazy stop and go approach, while avoiding aborting transactions, can introduce a long blocking time (which, naturally, gets worse for scenarios where transactions have a longer duration that can go up to 10000ms). In conclusion, the eager stop and go trades a lower stopping time for a high abort rate, which can result in the abort of 80% of the transactions during switch time.

Let us now consider the fast switching algorithm for commuting from 2PC to TO (Alg. 3), whose performance is evaluated in the Figure 8. In this fast switch algorithm nodes must first wait for all local pending transactions initiated with 2PC to terminate before accepting transactions to be processed by TO. Therefore, this algorithm also introduces some amount of blocking time that, although smaller than in the case of the stop and go switching algorithm, is no longer negligible. Nevertheless, the advantages of fast switching are still very significant when transactions are very long since its blocking time does not depend of the transactions' duration (10000 ms vs. 0.1ms). The eager version of stop and go still provides a constant low blocking time (10ms), but at the cost of aborting on average 80% of the transactions during switch time.

These results show that, whenever available, the use of specialized fast switching algorithms is preferable. On the other hand, the stop and go algorithm can be implemented without any knowledge about the semantics of the replication protocols. Also, the eager version can provide reasonably small blocking times (in the order of 10ms) at the cost of aborting some transactions during the reconfiguration.

6.3 Performance of MorphR

Figures 9(a) and 9(b) compare the throughput of MorphR with that of a statically configured, non-adaptive version of In-



(a) Blocking time. (b) Aborts.

Fig. 7. Switch between 2PC and PB.

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

finispan. In addition to the real-time performance of MORPHR, we also report the average performance achieved by the three baseline protocols for all the presented workloads. The models were trained with the previously presented Radargun and TPC-C datasets, from which we removed the workloads RW1-3 and TW1-3 reported in Table 1. In this experiment, we injected load in the system for a duration of 6 minutes and configured the RPSO to query the system state every 30 seconds to predict the protocol to be used. The plots show that, whenever the workload changes, the RPSO detects it and promptly switches to the most appropriate protocol. As expected, the performance of MORPHR keeps up with that of the best static configuration, independently of the benchmark. We can also observe that the overheads introduced by the supports for adaptivity are very reduced given that, when MORPHR stabilizes, its throughput is very close to one achieved when using a static configuration.

Figure 9(c) evaluates the effectiveness of the MORPHR with Geograph. The models were trained using various Geograph workloads containing a varying percentage of agent types. The RSPO was configured to query the system every two minutes. We injected a variable workload in which the three workloads reported in Table 1, GW1-3, take place sequentially, each one lasting 30 minutes. We configure Infinispan to operate initially with the PB protocol, which is suboptimal given that, as shown in Figure 1, the optimal protocol for GW1 is 2PC. The plot highlights that the change in the workload at minute 30 leads to a drastic degradation of performance when using 2PC due to the very high data contention level of GW2. MORPHR switches to PB, the optimal protocol for this workload, two minutes later. At minute 60 the workload GW3 is triggered, and we can observe an increase in the throughput due to the decrease in the workload mix of the percentage of transactions injected by Blogger agents, which are longer to process, more prone to conflict, and which generate larger commit messages than the other transaction classes. However, the optimal replication protocol for this workload is TO, which, despite generating higher contention and network load than PB, lets all nodes process update transactions. This opportunity is correctly identified by the RPSO, which triggers the switch towards TO at around minute 62, ensuring, again, the optimality of the configuration.

7 RELATED WORK

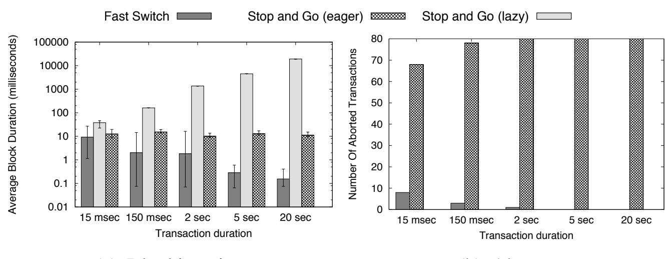
We classify related work into the following categories: i) work on protocol reconfiguration in general; ii) work on automated resource provisioning; iii) work on self-tuning in databases systems, both distributed and centralized; iv) work on adaptive STMs; and v) decision making in adaptive systems. We will address each of these categories in turn.

An extensive set of works has been produced on dynamic protocol reconfiguration [26], [27], [28]. A large part of this work has focused on the reconfiguration of communication protocols. For instance, the work in [28] proposes an Atomic Broadcast (AB) generic meta-protocol that allows to stop an executing instance of an AB protocol, and to activate a new one. Our work encompasses a larger stack of software layers, and takes into account the inter-dependencies between the replica control and concurrency control schemes. Also, in MORPHR we address also the issue of how to automatically determine *when* to trigger adaptation, and not only *how*.

The area of automated resource provisioning is related to this work as it aims at reacting to changes in the workload and access patterns to autonomically adapt the system's resources. Examples include works in both transactional [13], [29] and non-transactional domains, such as Map-Reduce [30], [31] and VM sizing [32]. Analogously to MORPHR, several of these systems also use machine-learning techniques to drive the adaptation. However, the problem of reconfiguring the replication protocol raises additional challenges, e.g. by demanding dedicated schemes to enforce consistency during the transitioning between two replication strategies.

To the best of our knowledge, the work in [33] pioneers the issues associated with adaptation in transactional systems (specifically, DBMSs). In particular, this paper identifies a set of sufficient conditions for supporting non-blocking switches between concurrency control protocols. However, in order to satisfy them it is necessary to enforce very stringent assumptions (such as knowing a-priori whether the transactions will exhibit any data dependency). Our solution, on the other hand, relies on a framework that supports switching between generic replication protocols without requiring any assumption on the workload generated by applications. Several other approaches have also been proposed based on the idea to automatically analyse the incoming workload, e.g. [34], to automatically identify the optimal database physical design or self-tune some of the inner management schemes, e.g. [35]. However, none of these approaches investigated the issues related to adapt the replication scheme. Even though the work in [36] presents a meta-protocol for switching between replication schemes, it does not provide a mechanism to autonomically determine the most appropriate scheme for the current conditions.

A number of works have been aimed at automatically tuning the performance of Software Transactional Memory (STM) systems, even if most of these works do not consider replicated systems. In [37], the authors present a framework for automatically tuning the performance of the system by switching between different STM algorithms. This work was based in RSTM [38], which allows changing both STM algorithms and configuration parameters within the same algorithm. The main difference between RSTM and our work is that the latter system must stop processing transactions whenever changing the (local) concurrency control algorithm, whereas MORPHR provides mechanisms allowing the coexistence of protocols

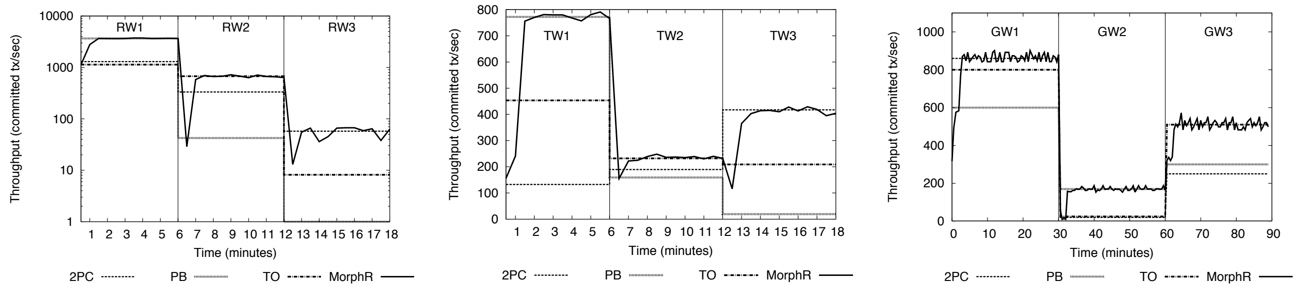


(a) Blocking time. (b) Aborts.

Fig. 8. Switch between 2PC and TO.

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460



(a) Radargun. (b) TPC-C. (c) Geograph.

Fig. 9. MORPHR performance vs static configurations.

while the switch is in process. The works in [39] and [40] also allow changing configuration parameters, but our framework targets the problem of changing the protocol as a whole. Other works addressing adaptability in transaction processing include [41], [42] but they tackle different issues non-related to the reconfiguration of replication protocols.

Hybrid Transactional Replication [43] proposes an hybrid scheme allowing concurrent transactions to execute using the state machine replication or a deferred update approach. Contrarily to MORPHR's generic approach, this hybrid replication scheme supports switching exclusively between these two specific approaches. Also, this work does not address the problem of how to determine which replication protocol to use when faced with a given workload.

Our previous work, PolyCert [44], uses ML techniques [45] to determine which is the most appropriate replication protocol according to each transaction's characteristics for in-memory transactional data grids. However, in Polycert it is only possible to use protocols from the same family, namely certification based replication protocols, which only differ in the way transactions are validated. In this work, we address the more complex and generic problem of adaptive reconfiguration among arbitrary replication schemes.

MORPHR is also related to the body of research on the specification of adaptation policies [46]. These approaches provide an infrastructure that allows experts (such as programmers or system administrators) to control the adaptation policies of a complex system by means of different types of rules' systems. Our approach, however, relies on ML techniques to automate the determination of the adaptation policy (an idea already explored in different contexts [13], [32], [47]). Also, it adopts a generic software architecture that promotes extensibility/development of specialized protocols, and support arbitrary adaptations in efficient ways.

Finally, recent middleware for building distributed applications, such as Sinfonia [48] and Chubby [49], also uses consensus as a building block. In the literature, a number of alternative consensus implementations have been proposed, optimized for different workloads, platform's scale etc. Therefore the ideas and techniques presented in this paper could also be applied to these systems.

8 CONCLUSIONS

This paper has presented MORPHR, a framework aimed to automatically adapt the replication protocol of in-memory transactional platforms according to the current operational conditions. MORPHR uses a modular approach supporting both general-purpose switching strategies, as well as optimized fast switch algorithms that can support non-blocking reconfiguration. We modelled the problem of identifying the optimal replication protocol given the current workload as a classification problem, and exploited several ML techniques to drive adaptation. MORPHR has been implemented in a well-known open source transactional data grid and extensively evaluated, demonstrating its high accuracy in the identification of the optimal replication strategy and the minimal overheads introduced to support adaptability.

REFERENCES

[1] M. Stonebraker, S. Madden, D. Abadi, S. Harizopoulos, N. Hachem, and P. Helland, "The end of an architectural era: (it's time for a complete rewrite)," in *Proc. VLDB 2007*. VLDB Endowment, 2007, pp. 1150–1160.

[2] K. Manassiev, M. Mihailescu, and C. Amza, "Exploiting distributed version concurrency in a transactional memory cluster," in *Proc. PPoPP 2006*. ACM, 2006, pp. 198–208.

[3] M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues, "D2STM: Dependable distributed software transactional memory," in *Proc. PRDC 2009*, 2009, pp. 307–313.

[4] F. Pedone, R. Guerraoui, and A. Schiper, "The database state machine approach," *Distributed and Parallel Databases*, vol. 14, no. 1, pp. 71–98, 2003.

[5] B. Kemme and G. Alonso, "Don't be lazy, be consistent: Postgres-r, a new way to implement database replication," in *Proc. VLDB 2000*. Morgan Kaufmann Publishers Inc., 2000, pp. 134–143.

[6] N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg, *The primary-backup approach*. ACM Press/Addison-Wesley, 1993, pp. 199–216.

[7] P. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1986.

[8] F. Marchioni, *Infinispan Data Grid Platform*. Packt Publishing, Limited, 2012. [Online]. Available: http://www.google.pt/books?id=h0SQD3TTuE8C

[9] ISO, *ISO/IEC 9075-2:1999: Information technology — Database languages — SQL — Part 2: Foundation (SQL/Foundation)*. International Organization for Standardization, 1999.

[10] X. Defago, A. Schiper, and P. Urban, "Total order broadcast and multicast algorithms: Taxonomy and survey," *ACM Computing Surveys*, vol. 36, no. 4, pp. 372–421, 2004.

[11] F. Raab, "TPC-C-the standard benchmark for online transaction processing (OLTP)." 1993.

[12] V. Ziparo, F. Cottefoglie, D. Calisi, M. Zaratti, F. Giannone, and P. Romano, "D4.3 - prototype of pilot application i," in *Cloud-TM project*, 2013. [Online]. Available: http://cloudtm.ist.utl.pt/

[13] D. Didona, P. Romano, S. Peluso, and F. Quaglia, "Transactional auto scaler: Elastic scaling of in-memory transactional data grids," in *Proc. ICAC 2012*, 2012.

[14] T. Mitchell, *Machine learning*, ser. McGraw Hill series in computer science. McGraw-Hill, 1997.

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.

This is the author's version of an article that has been published in this journal. Changes were made to this version by the publisher prior to publication.
The final version of record is available at http://dx.doi.org/10.1109/TPDS.2014.2363460

[15] K. Singh, E. İpek, S. McKee, B. R. de Supinski, M. Schulz, and R. Caruana, "Predicting parallel application performance via machine learning approaches: Research articles," *Concurr. Comput. : Pract. Exper.*, vol. 19, no. 17, pp. 2219–2235, Dec. 2007.
[16] J. Quinlan, "C5.0/see5.0," http://www.rulequest.com/see5-info.html.
[17] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1994.
[18] J. Platt, "Advances in kernel methods," B. Schölkopf, C. J. C. Burges, and A. J. Smola, Eds. MIT Press, 1999, ch. Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pp. 185–208.
[19] E. Frank, M. Hall, G. Holmes, R. Kirkby, B. Pfahringer, and I. Witten, *Weka: A machine learning workbench for data mining*. Springer, 2005, pp. 1305–1314.
[20] D. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin, "Backpropagation," Y. Chauvin and D. E. Rumelhart, Eds. L. Erlbaum Associates Inc., 1995, ch. Backpropagation: The Basic Theory, pp. 1–34.
[21] S. Keerthi, S. Shevade, C. Bhattacharyya, and K. Murthy, "Improvements to platt's smo algorithm for SVM classifier design," *Neural Comput.*, vol. 13, no. 3, pp. 637–649, Mar. 2001.
[22] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artif. Intell. Rev.*, vol. 22, no. 2, pp. 85–126, Oct. 2004.
[23] E. Page, "Continuous inspection schemes," *Biometrika*, pp. 100–115, 1954.
[24] R. Kalman, "A new approach to linear filtering and prediction problems," *J. Fluids Eng.*, vol. 82, no. 1, pp. 35–45, 1960.
[25] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *The J. of Machine Learning Research*, vol. 3, pp. 1157–1182, 2003.
[26] X. Liu, R. van Renesse, M. Bickford, C. Kreitz, and R. Constable, "Protocol switching: Exploiting meta-properties," in *Proc.WARGC 2001*, 2001, pp. 37–42.
[27] W. Chen, M. Hiltunen, and R. Schlichting, "Constructing adaptive software in distributed systems," in *Proc. ICDCS 2001*. IEEE Computer Society, 2001, pp. 635–643.
[28] O. Rütti, P. Wojciechowski, and A. Schiper, "Structural and algorithmic issues of dynamic protocol update," in *Proc. IPDPS 2006*. IEEE Computer Society, 2006, pp. 133–133.
[29] P. Xiong, Y. Chi, S. Zhu, J. Tatemura, C. Pu, and H. HacigümüŞ, "Activesla: a profit-oriented admission control framework for database-as-a-service providers," in *Proc. SOCC 2011*, 2011, pp. 15:1–15:14.
[30] H. Herodotou, F. Dong, and S. Babu, "No one (cluster) size fits all: automatic cluster sizing for data-intensive analytics," in *Proc. SOCC 2011*. ACM, 2011, pp. 18:1–18:14.
[31] N. Yigitbasi, T. L. Willke, G. Liao, and D. Epema, "Towards machine learning-based auto-tuning of mapreduce," in *Proc. MASCOTS 2013*. IEEE Computer Society, 2013, pp. 11–20.
[32] L. Wang, J. Xu, M. Zhao, Y. Tu, and J. Fortes, "Fuzzy modeling based resource management for virtualized database systems," in *Proc. MASCOTS 2011*. IEEE Computer Society, 2011, pp. 32–42.
[33] B. Bhargava and J. Riedl, "A model for adaptable systems for transaction processing," *IEEE TKDE*, vol. 1, no. 4, pp. 433–449, Dec. 1989.
[34] P. Martin, S. Elnaffar, and T. Wasserman, "Workload models for autonomic database management systems," in *Proc. ICAS 2006*. IEEE Computer Society, 2006, p. 10.
[35] N. Bruno and S. Chaudhuri, "An online approach to physical design tuning," in *Proc. ICDE 2007*, 2007, pp. 826–835.
[36] M. Ruiz-Fuertes and F. Munoz-Escoi, "Performance evaluation of a metaprotocol for database replication adaptability," in *Proc. SRDS 2009*. IEEE Computer Society, 2009, pp. 32–38.
[37] Q. Wang, S. Kulkarni, J. Cavazos, and M. Spear, "A transactional memory with automatic performance tuning," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, pp. 54:1–54:23, Jan. 2012.
[38] M. Spear, "Lightweight, robust adaptivity for software transactional memory," in *Proc. SPAA 2010*. ACM, 2010, pp. 273–283.
[39] P. Felber, C. Fetzer, and T. Riegel, "Dynamic performance tuning of word-based software transactional memory," in *Proc. PPoPP 2008*. ACM, 2008, pp. 237–246.
[40] V. Marathe, W. Scherer, and M. Scott, "Adaptive software transactional memory," in *Distributed Computing*, 2005, pp. 354–368.
[41] M. Holanda, A. Brayner, and S. Fialho, "Introducing self-adaptability into transaction processing," in *Proc. SAC*, 2008, pp. 992–997.
[42] M. Castro, L. Goes, C. Ribeiro, M. Cole, M. Cintra, and J.-F. Mehaut, "A machine learning-based approach for thread mapping on transactional memory applications," in *Proc. HiPC 2011*, dec. 2011, pp. 1 –10.
[43] T. Kobus, M. Kokocinski, and P. Wojciechowski, "Hybrid replication: State-machine-based and deferred-update replication schemes combined," in *Proc. ICDCS 2013*, 2013.
[44] M. Couceiro, P. Romano, and L. Rodrigues, "Polycert: Polymorphic self-optimizing replication for in-memory transactional grids," in *Proc. Middleware 2011*. Springer Berlin / Heidelberg, 2011, pp. 309–328.
[45] ——, "A machine learning approach to performance prediction of total order broadcast protocols," in *Proc. SASO 2010*, 2010, pp. 184 –193.
[46] L. Rosa, L. Rodrigues, A. Lopes, M. Hiltunen, and R. Schlichting, "Self-management of adaptable component-based applications," *IEEE Trans. Softw. Eng.*, vol. 39, no. 3, pp. 403–421, Mar. 2013.
[47] R. Nathuji, A. Kansal, and A. Ghaffarkhah, "Q-clouds: Managing performance interference effects for qos-aware clouds," in *Proc. Eurosys 2010*. ACM, 2010, pp. 237–250.
[48] M. Aguilera, A. Merchant, M. Shah, A. Veitch, and C. Karamanolis, "Sinfonia: A new paradigm for building scalable distributed systems," in *Proc. SOSP 2007*. ACM, 2007, pp. 159–174.
[49] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," in *Proc. OSDI 2006*. USENIX Association, 2006, pp. 335–350.

Maria Couceiro Maria Couceiro is a Ph.D. candidate in Information Systems and Computer Engineering at Instituto Superior Técnico in Portugal, and a researcher at the Distributed Systems Group of INESC-ID Lisboa. She received a MSc in Information Systems and Computer Engineering from Instituto Superior Técnico in 2009 and currently works in distributed transactional memory, adaptive systems and autonomic computing.

Pedro Ruivo Pedro Ruivo has an MSc degree in Information Systems and Computer Engineering at Instituto Superior Técnico in Portugal. He is currently working for Red Hat Inc. on the Infinispan project. His main interests lie in the areas of distributed key-value stores and transactional memory.

Paolo Romano Paolo Romano is a senior researcher at INESC-ID since 2008. In 2012 he joined Instituto Superior Técnico in Lisbon as an assistant professor. His research interests include distributed computing, dependability, autonomic systems, performance modelling, and cloud computing. He has published more than 90 papers and serves regularly as Program Committee member and reviewer for prestigious international conferences and journals.

Luis Rodrigues Luis Rodrigues is a professor at Instituto Superior Técnico, Universidade de Lisboa and a researcher at INESC-ID where he now serves in the board of directors. His research interests lie in the area of reliable distributed systems. He is the co-author of more than 150 papers and 3 textbooks on these topics.

Copyright (c) 2014 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.