Exploiting Total Order Multicast in Weakly Consistent Transactional Caches

Pedro Ruivo, Maria Couceiro, Paolo Romano and Luís Rodrigues
INESC-ID Lisboa, Instituto Superior Técnico, Universidade Técnica de Lisboa
Email: {pruivo, mcouceiro, romanop}@gsd.inesc-id.pt, ler@ist.utl.pt

***Abstract*—Nowadays, distributed in-memory caches are increasingly used as a way to improve the performance of applications that require frequent access to large amounts of data. In order to maximize performance and scalability, these platforms typically rely on weakly consistent partial replication mechanisms. These schemes partition the data across the nodes and ensure a predefined (and typically very small) replication degree, thus maximizing the global memory capacity of the platform and ensuring that the cost to ensure replica consistency remains constant as the scale of the platform grows. Moreover, even though several of these platforms provide transactional support, they typically sacrifice consistency, ensuring guarantees that are weaker than classic 1-copy serializability, but that allow for more efficient implementations.**

This paper proposes and evaluates two partial replication techniques, providing different (weak) consistency guarantees, but having in common the reliance on total order multicast primitives to serialize transactions without incurring in distributed deadlocks, a main source of inefficiency of classical two-phase commit (2PC) based replication mechanisms.

We integrate the proposed replication schemes into Infinispan, a prominent open-source distributed in-memory cache, which represents the reference clustering solution for the well-known JBoss AS platform. Our performance evaluation highlights speed-ups of up to 40x when using the proposed algorithms with respect to the native Infinispan replication mechanism, which relies on classic 2PC-based replication.

***Keywords*-Partial Replication, Distributed Memory, Transactional Memory, Atomic Multicast**

I. Introduction

Nowadays, distributed in-memory caches are increasingly used to improve the performance of applications that require frequent access to large amounts of data, by decoupling the persistent memory access from the critical path of the application. YouTube, Wikipedia, Twitter and Facebook are a few examples of applications that make use of this architectural approach. The key reason underlying the success of these platforms lies in their ability to achieve higher performance, scalability and elasticity (namely, the ability to dynamically scale up or down the number of distributed physical nodes composing the platform), when compared to classical SQL-based database management systems. This is achieved thanks to the reliance on (i) simpler data models, e.g., key/value pairs vs. relational model, (ii) more efficient application interfaces, namely embedded vs. JDBC/ODBC connections, and (iii) the reliance on in-memory replication and asynchronous write to disk vs. (per-transaction) synchronous logging to disk.

In this context, data partitioning and in-memory replication across multiple distributed nodes has two main advantages: on one hand, it allows distributing load among multiple replicas, enhancing throughput; on the other hand, it ensures the survival of data if a replica fails. This last point is particularly relevant, since the data is first stored in volatile memory and made persistent asynchronously (and would therefore be lost in case of failure, if it was not replicated).

However, there are also costs inherent to replication that must not be overlooked. Firstly, replicas consume memory, reducing the amount of information that can be stored in the cache. Also, the larger the number of replicas, the more expensive it becomes to ensure their consistency.

Partial replication tries to overcome these disadvantages by configuring the cache in such a way that each item is replicated in a subset of nodes and no node stores all the data. This paper studies the use of partial replication techniques in the context of in-memory caches. Even though partial replication has already been applied to distributed databases [1], there are significant differences in the workloads imposed to both systems and on the kind of processing related to the execution of transactions in both scenarios. More specifically, database management systems have the additional overheads of SQL parsing, synchronous storage, etc., that are absent in distributed in-memory cache systems [2]. Consequently, the coordination costs associated with partial replication in the context of in-memory caches are amplified. Due to these reasons, several of the mainstream in-memory cache platforms have opted for relaxing consistency, ensuring weaker semantics than the classical 1-copy seriliazability [3] in order to allow more efficient implementations.

In this paper we present two replication algorithms, ensuring different weak consistency criteria, but both relying on the usage of Total Order Multicast (TOM) [4] to ensure agreement on the transaction serialization order in a genuine fashion [5], i.e. involving in the coordination for a transaction T only the nodes responsible for storing a copy of the data accessed by T. The proposed solutions are inspired by recent literature in the area of 1-copy serializable partial replication schemes [5], which we adapt in order to guarantee weaker consistency guarantees. Thanks

to their reliance on TOM, the proposed protocols avoid the occurrence of distributed deadlocks, which represent the key source of inefficiency in 2PC-based replication schemes [6].

We integrated the proposed weakly consistent partial replication protocols into one of the mainstream open-source distributed in-memory cache platforms, namely Red Hat's/JBoss' Infinispan. Infinispan is a complex, weakly consistent, in-memory transactional data grid that represents the reference solution to support clustering of the well-known JBoss AS (probably the most widely used Java application server at the time of writing [7]).

We perform an experimental evaluation study in which we compare the performance of the proposed partial replication schemes with those built-in into Infinispan, which rely on a classical Two Phase Commit (2PC) based replication scheme. We consider both synthetic workloads, which allow to assess the protocol performance in heterogeneous (and clearly identifiable) scenarios, and industry standard benchmarks for OLTP systems, namely the TPC-C benchmark [8]. Our experimental study highlights that the proposed TOM-based schemes achieve striking speed-ups (up to 40x) with respect to classic 2PC-based solutions in high contention scenarios, while achieving very similar performance in presence of workloads with very limited contention.

The rest of the paper is organized as follows. Section II briefly describes Infinispan and how it manages replication and distribution. Section III presents the mechanisms of partial replication developed to enhance Infinispan. In Section IV the performance of the proposed system is evaluated. Section V compares our solution with related work. Finally, Section VI concludes this paper.

II. INFINISPAN

Infinispan [9] is a popular open source in-memory distributed transactional cache developed by JBoss, Red Hat. Analogously to many other contemporary distributed cache platforms, Infinispan externalizes a simple key/value store interface (via the standard JSR-107 [10] JCache interface), providing support for transactions and for two main operational modes: partial vs. full data replication (referred to as *distribution* vs. *replication* modes in Infinispan), depending on whether the data (i.e. key/value pairs) is replicated on a subset or on the whole set of nodes in the data grid. As already mentioned, in this paper we focus on the partial replication mode.

In the partial replication mode, Infinispan relies on a lightweight consistent hashing scheme [11] to partition data across replicas, ensuring good load balancing (in terms of number of keys hosted by each replica) and minimum reshuffling of keys in presence of joins/departures of nodes from the platform. Also, it supports replication of each key across a fixed, user-tunable number of replicas, achieving fault-tolerance without hampering scalability (unlike full replication schemes).

Infinispan supports transactions in a weak-consistent flavour, opting for more relaxed criteria than the classic 1-copy serializability [3]. Specifically, Infinispan supports the following (weaker) consistency criteria [12]:

- **Read Committed (RC)** which ensures that a transaction can only read previously committed values.
- **Repeatable Read (RR)** which ensures that no two consecutive reads within the same transaction can return different values.
- **Repeatable Read with Write Skew Check (RR+WS)** which, in addition to ensure that no two consecutive reads within the same transaction can return different values, checks also if the value of a key was changed between a consecutive read and write operation of a transaction (aborting, in such a case, the transaction).

The choice to support weak (i.e. non-serializable) consistency criteria has had clearly an impact on the design of both the local concurrency control mechanism, and on the replica coordination protocol. More in detail, Infinispan implements a lightweight, non-serializable variant of the multi-version concurrency control algorithm which never blocks or aborts a transaction during a read operation, and relies on an encounter-time locking strategy to detect write-write conflicts.

More in detail, for what concerns read operations, if the RC consistency criterion is being used, Infinispan simply returns the latest committed value. If, instead, RR is being used, whenever a transaction issues a read on a data item, it stores the returned value into its transactional context, and returns it in subsequent read operations. It is important to point out that the data is distributed, so read operations may require contacting other nodes (even though Infinispan tries to reduce the frequency of remote read operations by adopting an additional, so called, L1 cache [9]).

Write operations, on the other hand, do not require distributed interaction during transaction execution. Instead, whenever a key/value pair is updated/inserted/deleted (simply referred to as write operation in the following), the lock on the corresponding key is acquired locally during the transaction execution phase. If the write skew check is enabled (namely, the RR+WS consistency criteria are being used), however, a further check is performed upon issuing of a write operation: if the transaction had previously read that key, and its value is found to be different after having acquired the corresponding lock, the transaction is simply aborted.

Since objects are distributed through several nodes, it is necessary to guarantee that the updates made by a transaction are applied atomically at all replicas or at none of them. Infinispan achieves this by employing the classic Two Phase Commit protocol [13] (2PC). During the first phase of 2PC, the nodes attempt to remotely acquire the locks on all the replicas that are responsible for storing the data updated during the local execution of the transaction.

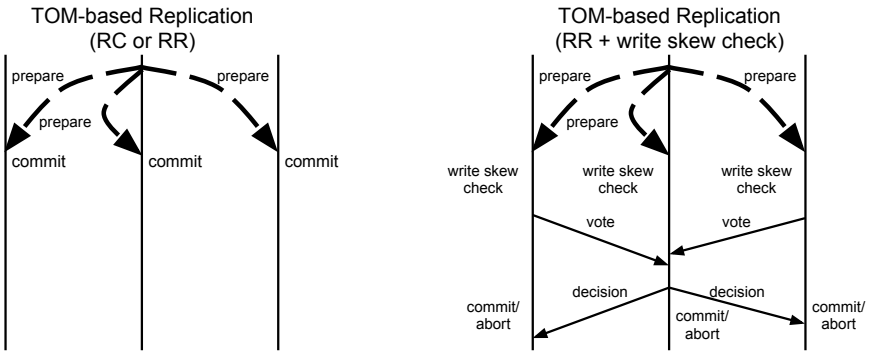


Figure 1: Proposed Total Order Multicast-based Replication Algorithms.

Replica location is determined by a combination of the group membership services provided by JGroups [14], along with the consistent hashing mechanism mentioned above.

If the lock acquisition phase succeeds on all the contacted replicas, the transaction originator finally sends a commit message and commits locally. In presence of conflicting, concurrent transactions, however, the lock acquisition phase may fail due to the occurrence of distributed deadlocks. In Infinispan deadlocks are detected using a simple timeout based approach, which uses conservative values to minimize the risk of false-positives (the default value is set to 10 seconds), and is coupled with an eager deadlock detection algorithm that detects circular, direct lock waits between two transactions (thus not detecting deadlocks due to chains of more than two transactions in circular transitive wait); Infinispan resolves the deadlock deterministically by aborting one of the blocked transactions. If the lock acquisition fails during the prepare phase, a negative vote is sent to the coordinator, which, in turn, instructs all replicas to abort the transaction.

III. TOM-based Partial Replication

As described in the previous section, Infinispan has already a built-in support for partial replication. Unfortunately, the 2PC-based replication scheme used by Infinispan is known to be prone to thrashing at non-minimal contention levels [6] due to the occurrence of distributed deadlocks (as it will also be shown in the following).

In this section we present two TOM-based partial replication algorithms. These algorithms ensure the same weak consistency criteria currently supported by Infinispan, but, not incurring in distributed deadlocks, they can sustain much higher throughputs (committed transactions per second) especially in scenarios of moderate/high contention.

In the following, we first introduce the two TOM-based replication algorithms, and then discuss two alternative implementations of the TOM primitive, exhibiting a trade-off between the number communication steps and message complexity.

A. TOM-based partial replication schemes

Both the partial replication algorithms presented in this paper rely on the same base principle: using TOM to achieve agreement among all replicas whose keys have been updated by a committing transaction T on T's serialization order.

As in Infinispan's baseline algorithm, in fact, transactions execute locally (with the exception of remote read operations, which may require to fetch data from remote nodes) until they enter their commit phase. At this stage, the entries updated by transactions (along with their previous values, in case of RR+WS consistency) are sent, using total order multicast [4], to all the replicas that need to be updated. This set of replicas is given by the union of the replicas maintaining a copy of each of the objects modified by the transaction and is, typically, only a subset of the number of nodes that compose the system.

By relying on a TOM primitive to disseminate the above messages, we can guarantee that if two replicas deliver two updates, they do it in the exact same order. Therefore, if the consistency criteria is either RC or RR, see Figure 1, the replicas can immediately apply the updates in the order in which they are delivered by the TOM primitive. This is sufficient to guarantee that all replicas apply the updates generated by all conflicting transactions in the same order, and is achieved in our implementation by having a single dedicated thread, which is awakened whenever a transaction is TOM-delivered, and is in charge of performing the write-back phase of both local and remote transactions.

On the other hand, see Figure 1, if the consistency criterion in use is RR+WS, upon TOM-deliver of a transaction, replicas need to perform the write-skew check in order to determine the transaction's outcome. This implies the need for the replicas to undergo an extra voting phase, during which each replica performs the write skew (on the keys of which is responsible) and sends back the result to the replica that executed the transaction. Before sending the final commit/ abort message, the replica that executed the transaction needs to wait until it is informed of the successful outcome of the write-skew check of all the updated keys.

Note that, since all the replicas that are responsible for the same set of keys certify the transaction deterministically and in the same order, it is guaranteed that they all determine the same outcome for the transaction. Therefore, in order to commit a transaction, the transaction coordinator does not need to wait for positive replies from all the nodes that it had contacted via the TOM primitive, but only until it receives a positive vote for each updated key from *at least* one of the nodes over which the key is replicated. As in classic 2PC, instead, the transaction is aborted as soon a negative vote message is received.

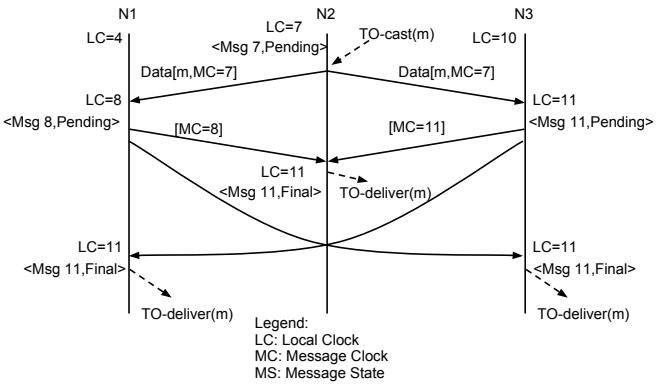
B. Implementing Total Order Multicast

We now address the implementation of the Total Order Multicast (TOM) primitive [4] used in the algorithms above. Informally, the TOM primitive allows disseminating a message m to a subgroup of the system nodes, denoted as $m.dst$, while ensuring agreement on the (total) order of delivery of messages in presence of i) concurrent TOMs triggered by different senders, ii) possible overlaps among the recipient sites of two TOMs, and iii) crashes of (a subset of) sites.

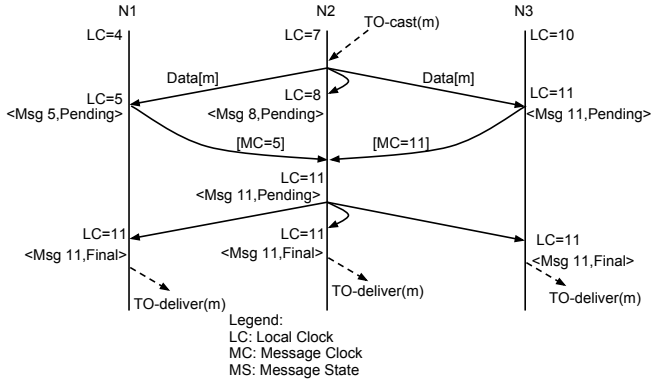
Formally, the TOM primitive guarantees the following properties: (i) *uniform integrity*: for any site s and any message m, s delivers m at most once, and only if $s \in m.dst$; (ii) *validity*: if a correct site s issues a TOM for a message m, then eventually all correct sites in $m.dst$ delivers m; (iii) *uniform agreement*: if site s delivers a message m, then eventually all correct sites in $m.dst$ deliver m; (iv) *uniform prefix order*: for any two messages m and m', and any two sites s and s', such that $\{s, s'\} \subseteq m.dest \cap m'.dest$, if s delivers m and s' delivers m', then either s delivers m' before m or s' delivers m before m'; (v) *uniform acyclic order*: the relation $<$ is acyclic, where $<$ is defined as follows: $m' < m$ if and only if any process delivers m and m' in that order.

A simple way to achieve total order in messages to different groups is simulating the multicast by sending all messages in total order to all replicas (in other words, to a single super-group, which is the union of all possible sub-groups). Afterwards, the replicas which are not the recipients of the messages discard them. This is clearly an inefficient approach as if forces all replicas to participate in all transactions.

In order to overcome this issue, we developed a selective total order multicast protocol for JGroups, inspired by the Skeen's algorithm described in [15] and used in an early version of the ISIS toolkit [4], which operates as follows (see Figure 2b). Each machine has a logical clock that is incremented when DATA or ORDER messages are received. The ordered multicast starts when a DATA message is sent to the group of replicas that participate in the transaction. When this message is received, each replica increments its logical clock, assigns the resulting timestamp to the message, changes the message's state to *Pending* and puts



(a) 2 phases



(b) 3 phases

Figure 2: Total Order Multicast Algorithms

it in an ordered queue; the local order number is then sent to the replica that originally sent the message. The sender collects all the sequence numbers assigned by the other replicas, determines their maximum value, and sends it back to all replicas in an ORDER message. Upon receiving this ORDER message, each replica updates the order number of the corresponding message, changing the order in the queue if necessary, marks the message as *Final* and updates its own logical clock. Finally, messages are delivered to the application when its state is marked as *Final* and they are in the head of the queue (i.e., there are no messages marked as *Pending* or *Final* with a lower timestamp).

We have also developed another version of the total order multicast protocol in order to understand the trade off of having less communication steps but exchanging more messages in each round. Its mode of operation is very similar to the first version, except that, upon delivery of the DATA message, replicas send the ORDER message (which piggybacks their logical clock value) to *all* the replicas to which the message is being multicast. With this decentralized approach, a replica can mark a message as *Final*, as soon as it receives all the ORDER messages corresponding to a given DATA message.

This version requires one less communication step to complete the commit of a transaction, see Figure 2a. However, the number of point-to-point messages exchanged between replicas increases quadratically with the number of participating replicas. The next section compares the two approaches, identifying the strengths and weaknesses of each one.

IV. Evaluation

This section presents the experimental evaluation of the proposed algorithms implementing partial replication in in-memory transactional distributed caches. This evaluation is based on a prototype developed by extending the code of Infinispan and JGroups to implement the algorithms described above.

A. Experimental Settings

All tests were ran on a cluster with 10 machines, where each machine is equipped with two 2.13 GHz *Quad-Core Intel(R) Xeon(R) E5506* processors and 16 GB of RAM, running *Linux 2.6.32-33-server* and interconnected via a private Gigabit Ethernet. We integrated the proposed TOM-based replication solutions in Infinispan 5.0 and JGroups 2.12. In our experiments, we use a number of machines varying between 4 and 10, and three distinct configurations of Infinispan: the native configuration, based on the protocol with deferred updates and coordination using two phase commit, and the two versions of the configuration developed by us, consisting of the algorithm based on genuine total order multicast. The native configuration uses timers to acquire locks, which expire after 10 seconds, and uses the deadlock detection technique described in Section II to detect deadlocks due to circular waits between two transactions in a more timely fashion. In both configurations we used a replication degree of 2, which means that each key is stored in two machines. For what concerns JGroups, we configured it to use UDP and IP multicast at the transport layer.

We used two different benchmarks to evaluate the system, Radargun, a benchmark created by RedHat specifically for this type of caches, and TPC-C [8], a more complex and realistic benchmark.

With Radargun we can compare the performance of several distributed caches (such as Infinispan, Ehcache [16], Coherence [17], etc.), in different scenarios and, by imposing a fairly high load on the different nodes of the system, it allows us to assess the maximum throughput of each configuration. This benchmark was adapted so that we could define different workloads with different conflict rates (i.e., concurrent accesses to the same objects). This enables us to gain more control over the experiments and easily establish correlations between the conflict rate, deadlock situations and throughput of the different configurations.

The workload used in these tests was the following. The application executes as many transactions as possible for a period of 5 minutes, using 8 threads in each machine submitting concurrent transactions to the system. Each transactions is composed of 10 operations. On average, 10% of these operations are writes and there is always at least one write operation per transaction. This way, there are no read-only transactions in the workload, because they do not require replica synchronization. To simulate low contention scenarios, transactions access random objects from a set of 100.000 keys and to simulate high contention scenarios, transactions access random objects from a set of only 1.000 keys.

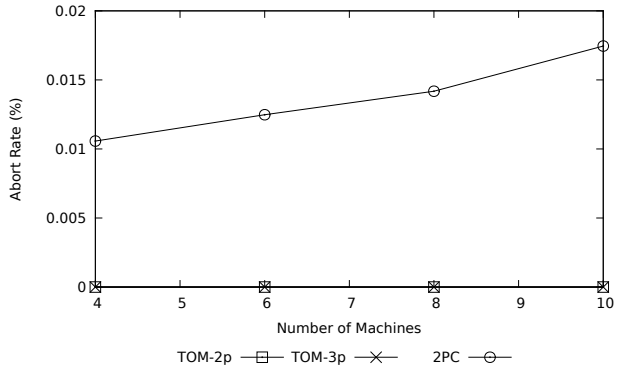
TPC-C is a benchmark that simulates a population of terminal operators executing transactions against a transactional data store. Transactions include entering and delivering orders, recording payments, checking the status of orders, and monitoring the level of stock at the warehouses. We adapted this benchmark so that only three types of operations are performed during its execution: entering new orders, querying the status of existing orders and entering payments from customers. The configuration we used is the following: one warehouse, 45% of payment transactions, 5% of query transactions (which means there will be 95% read-write transactions and only 5% of read only transactions) and 8 threads executing transactions on each machine.

B. Results

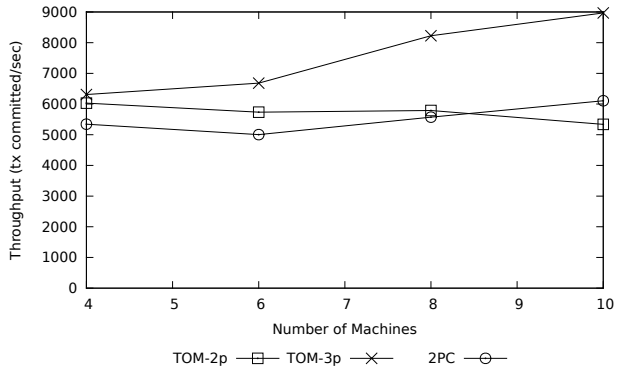
In the following we present a comparative evaluation of Infinispan's native solution (labelled as "2PC") and the two versions of our TOM implementation (labelled as "TOM-2p" and "TOM-3p" for the 2 and 3 communication step variants of total order multicast, respectively) using two consistency models: Read Committed and Repeatable Read, the latter with write skew anomaly detection enabled. The three performance metrics used in the plots are: abort rate, throughput of the system (committed transactions), and commit latency (time to complete the commit phase).

1) Abort Rate: Figures 3a, 4a, 5a and 6a depict the abort rate of the three algorithms in low and high contention scenarios for both consistency models for Radargun. As expected, for the TOM algorithms, the abort rate is virtually non-existent, even for the RR+WS model. On the other hand, the native algorithm needs to acquire locks in every participating replicas, causing deadlocks (and consequently transaction aborts) in case locks are acquired in different orders at different replicas. This issue becomes more noticeable as we increase the number of nodes in the system and, finally, in high contention scenarios, where the abort rate peaks at 3% for a system with 11 nodes.

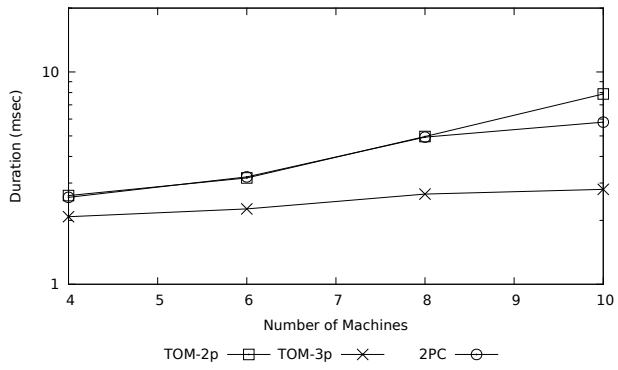
Figures 7a and 8a depict the abort rate for TPC-C. This benchmark induces a very high contention on a small subset of data items. Specifically, each write transaction must update one out of 10 existing instances of the, so called, *District* entity. Let us analyze first the Read Committed isolation level scenario. In this scenario, 2PC suffers from



(a) Abort Rate

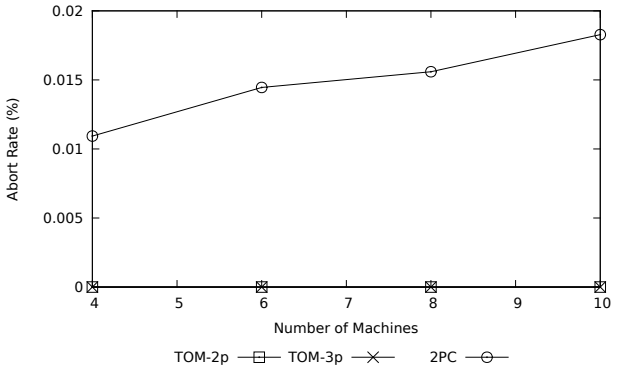


(b) Throughput

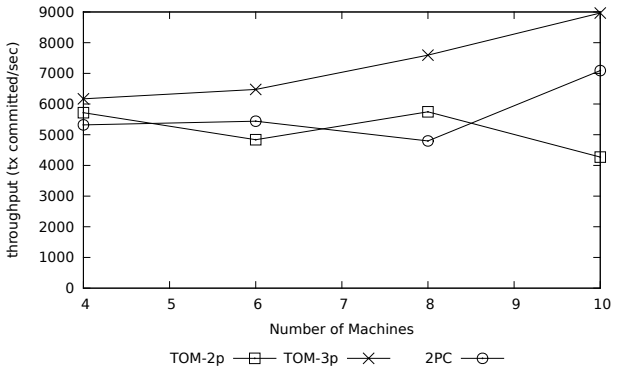


(c) Average Latency

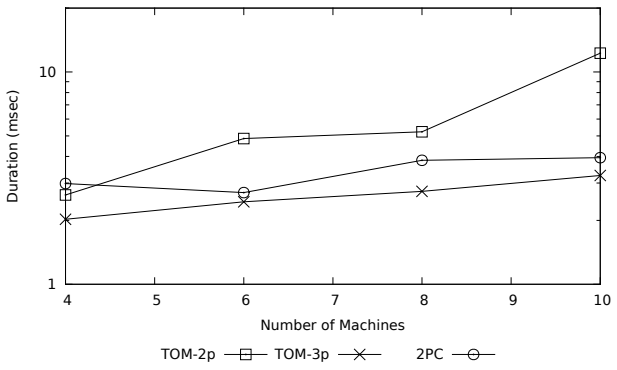
Figure 3: Radargun: Low Contention using Read Committed



(a) Abort Rate



(b) Throughput



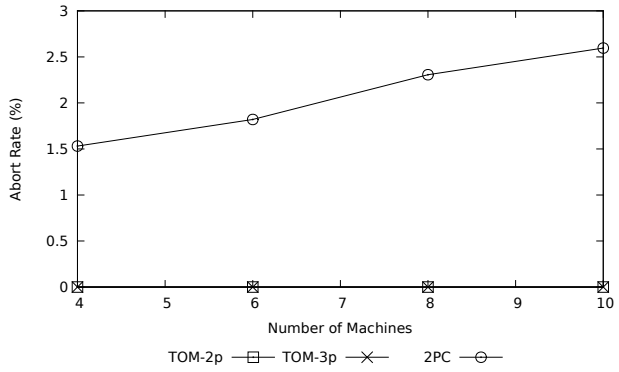
(c) Average Latency

Figure 4: Radargun: Low Contention using Repeatable Read with write skew detection

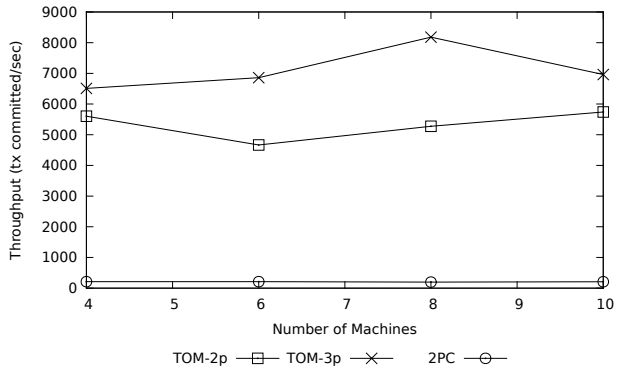
an abort rate ranging from 30% to 45%, which is entirely due to the occurrence of deadlocks. On the other hand, with the same isolation level, the TOM-based solutions do not suffer from any aborts.

When considering the Repeatable Read isolation level with write skew detection, we observe that the 2PC and the TOM-based solutions incurs on a very similar abort rate, ranging from 40% to 70%. This depends on the fact that TPC-C is very likely to generate read-write conflicts which, in turn, cause the failure of the write-skew test. Note that failure of the write-skew test represents the only abort cause for the TOM-based solutions. With 2PC, instead, only 27% of the transaction aborts are imputable to write-skew test failures, with (distributed) deadlocks being by far the most common cause of aborts.

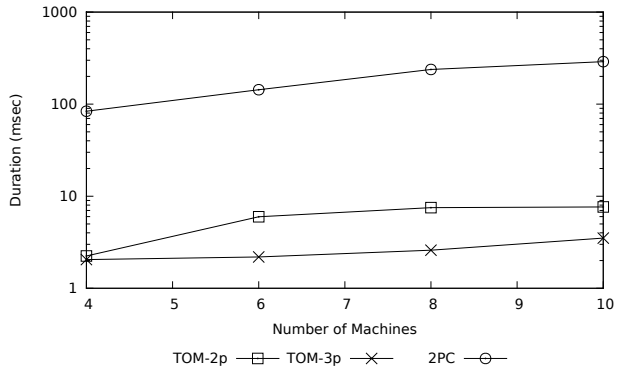
2) Throughput: Figures 3b, 4b, 5b and 6b present the effects of the abort rate on the throughput of the system using the Radargun benchmark, measured as the number of committed transactions per second. In low contention



(a) Abort Rate

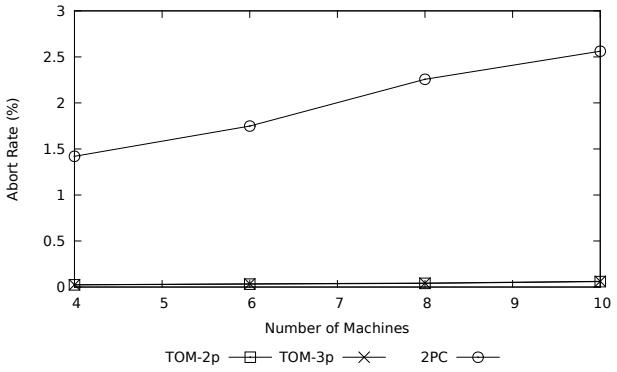


(b) Throughput

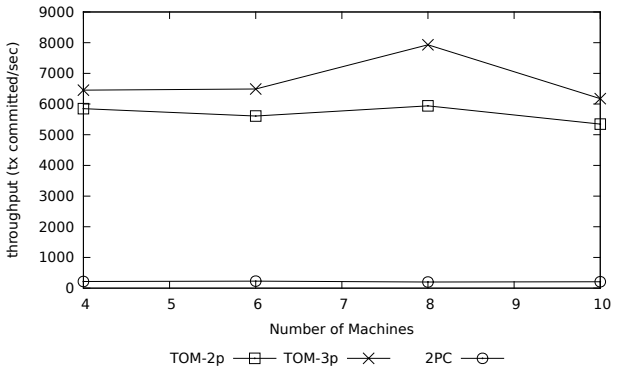


(c) Average Latency

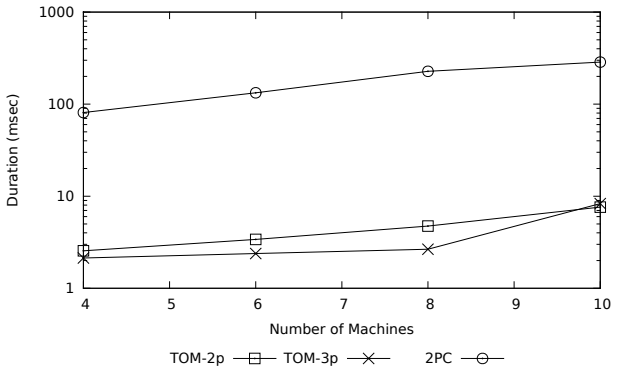
Figure 5: Radargun: High Contention using Read Committed



(a) Abort Rate



(b) Throughput



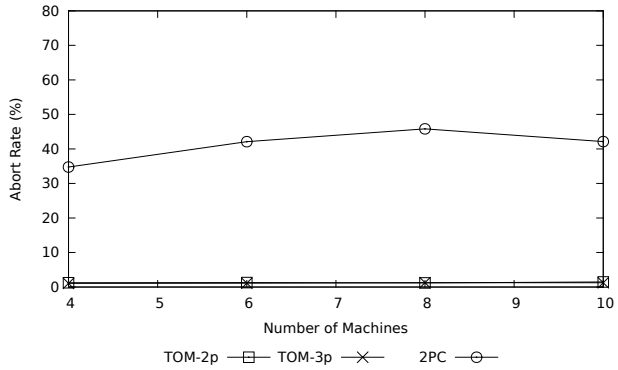
(c) Average Latency

Figure 6: Radargun: High Contention using Repeatable Read with write skew detection

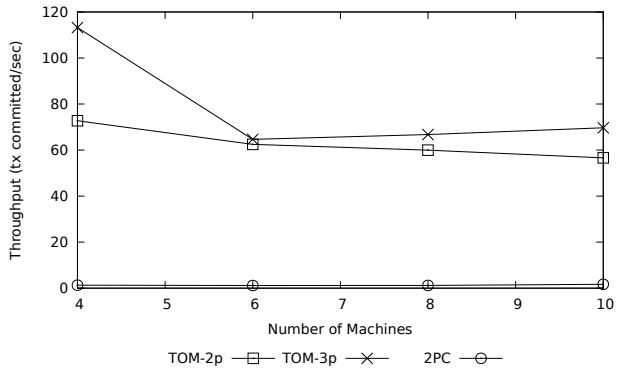
scenarios, which is the most favourable scenario for the 2PC protocol, 2PC, and TOM-2p have a similar throughput. However, TOM-3p has the best throughput. This is explainable by the fact that TOM-3p and 2PC have very similar communication patterns, but, unlike 2PC, TOM-3p does not incur in any deadlocks. The lower throughput for the TOM-2p is due to the high number of messages, which originate conflicts in the network and retransmissions, and additional processing load at the JGroups level. In the high contention scenario, however, 2PC's throughput is severely affected by the deadlocks, which results in our solutions delivering around 40 times higher throughput.

Figures 7b and 8b depict the throughput of the system using TPC-C. Due to the high contention generated by this benchmark, also in these scenarios, the TOM-based solutions achieve striking throughput gains with respect to 2PC, which thrashes due to the frequent occurrence of deadlocks.

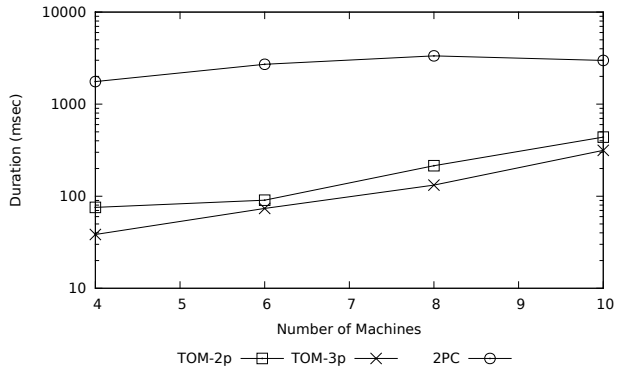
3) Latency: Figures 3c, 4c, 5c and 6c show the average latency of the commit phase for Radargun.



(a) Abort Rate

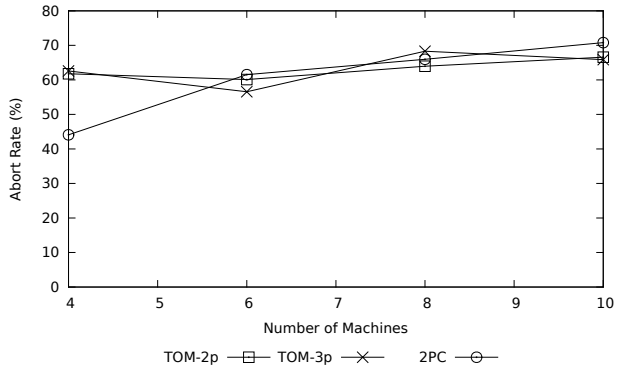


(b) Throughput

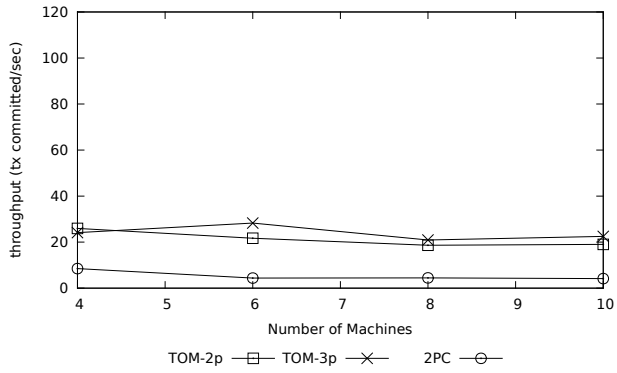


(c) Average Latency

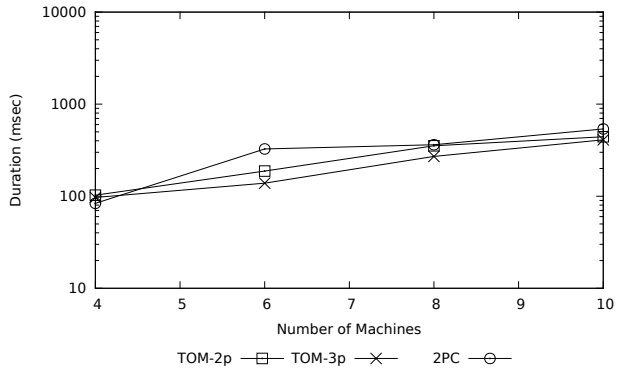
Figure 7: TPC-C: using Read Committed



(a) Abort Rate



(b) Throughput



(c) Average Latency

Figure 8: TPC-C: using Repeatable Read with write skew detection

With low contention, the performance of all algorithms is similar, due to the very low abort rate. In high contention scenarios, however, the commit phase latency for 2PC becomes up to 2 orders of magnitude higher than for the TOM-based solutions. This is explained considering that the commit latency includes also the time necessary to detect deadlocks occurring during the commit phase, and that these become very frequent at high contention levels when using 2PC. The plots highlight also that the latency for the TOM protocols is stable in both high and low contention scenarios, contrarily to what happens when using 2PC, whose performance is very dependent on the workload of the system. Analogous results are highlighted by Figures 7c and 8c, which show the latency of the commit phase for TPC-C. Also in this (high contention) scenario, the commit phase latency is significantly lower for the TOM-based protocols, being on average around one order of magnitude shorter than in the case of 2PC.

C. Discussion

Based on the previous results we can conclude that using TOM is beneficial in all the analysed scenarios. In addition, it is clear that the performance of 2PC is extremely dependent on the workload of the system; on the contrary, the performance of TOM is fairly stable both in low and high contention scenarios.

Interestingly, the TOM-3p solution outperforms the 2PC-based replication scheme even in the most favourable settings for 2PC, namely very low contention and RR+WS consistency model. We recall that, in these settings TOM-3p incurs in two additional communication steps with respect to 2PC. Also, the deadlock probability is below 0.02% with 2PC (being zero of course for the deadlock-free TOM-based solutions). Despite such a low deadlock probability, the large penalty affecting 2PC upon the occurrence of deadlocks has a non-negligible impact on 2PC performance, which result around 20% lower than for TOM-3p.

For what concerns TOM-2p, its overall performance is poorer than that of TOM-3p. Despite the fact that TOM-3p incurs in an additional communication step latency with respect to TOM-2p, our results highlight that the quadratic message complexity of TOM-2p leads in practice (at least in our experimental platform) to a detrimental effect on performance of the Group Communication System, which offsets the possible gains associated with the reduction in the number of communication steps.

With Radargun, the difference between RC and RR+WS consistency model is almost negligible. This is justified by the fact this is a synthetic benchmark in which the probability that the same key is read and written by the same transaction is quite low. Hence, the write skew mechanism is not activated frequently. However, in the TPC-C results it is possible to see the effects of this mechanism.

V. Related Work

This work results from the confluence of three different but closely related lines of investigation, namely: distributed in-memory cache systems, database replication techniques, and distributed transactional memory systems.

Distributed in-memory caches have emerged as tools to increase the performance of applications that require frequent low latency access to large amounts of data. The first proposed systems did not support transactions [18], [19], but more recent approaches have incorporated them in their architectures. Sinfonia [20] provides support for transactions and replication, but assumes that transactions are static, i.e., their read and write sets are known *a priori*. TxCache [21] relies on a back-end database to handle update transactions and ensures that read-only transactions users observe a strongly consistent snapshot of the cache (namely a consistent view of the system as of a specific timestamp). Conversely, our solution is designed to ensure weaker consistency criteria (read-committed with write-skew check being the strictest supported consistency criterion) and does not rely on any external transactional store to serialize update transactions.

The area of database replication is very rich in algorithms that ensure replica consistency in transactional environments. While most of these systems use full replication [22], [23], our focus is on those supporting partial replication [1], [24], [25]. P-Store,[5] is probably the solution that is closer in spirit to the approaches proposed in this paper. Also P-Store relies on a genuine algorithm, i.e., only the replicas involved in a given transaction participate in the coordination phase that ultimately leads to its commit or abort, and is built on top of a totally ordered multicast primitive. However, since P-Store's algorithm provides stronger consistency guarantees, it also incurs in additional costs (e.g. always requiring the certification of read-only transactions accessing data hosted by remote replicas, the dissemination of the whole transaction read-set during the commit phase, and a voting phase to determine the outcome of update transactions spanning multiple replica groups) when compared to the solutions proposed in this paper, which exploit a set of optimizations that are possible precisely because we target more relaxed consistency models.

Finally, distributed transactional memory systems [2] appeared as an extension to software transactional memory systems [26] developed for multi-core machines. The vast majority either does not consider fault-tolerance [27], [28], or are fully replicated [29], [30], [31].

VI. Conclusion

In this paper we presented a solution for supporting partial replication in distributed in-memory transactional caches. The proposed solution is inspired by algorithms developed in the context of database replication, and later adapted to support weaker consistency models. The result consists of a genuine partial replication algorithm (in which only the replicas of the data updated during a given transaction participate in its commit phase) which distributes the load of the system among its nodes. This solution was implemented in Infinispan and its performance was compared against the native support offered by the platform. Unlike the native solution, based on the two phase commit protocol, ours prevents deadlocks. The performance evaluation shows that the proposed solution, based on total order multicast, achieves a throughput up to forty times higher than the native one. As future work, we plan to extend our solution to support a stricter consistency model, namely, Serializability.

Acknowledgements

We are grateful to Sebastiano Peluso for porting the TPC-C benchmark to Infinispan. This work was partially supported by the European Commission through the Cloud-TM project (FP7-257784), and by FCT (INESC-ID multian-

nual funding) through the PIDDAC Program funds and the Aristos project (PTDC/EIA-EIA/102496/2008).

REFERENCES

[1] A. Sousa, F. Pedone, F. Moura, and R. Oliveira, "Partial Replication in the Database State Machine," in *Proc. of NCA*. IEEE CS, October 2001, pp. 298–309.

[2] P. Romano, N. Carvalho, and L. Rodrigues, "Towards distributed software transactional memory systems," in *Proc. of LADIS*. ACM, September 2008, pp. 4:1–4:4.

[3] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency control and recovery in database systems*. Addison-Wesley Longman Publishing Co., Inc., 1986.

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

[5] N. Schiper, P. Sutra, and F. Pedone, "P-store: Genuine partial replication in wide area networks," in *Proc of SRDS*. IEEE CS, November 2010, pp. 214–224.

[6] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *Proc. of SIGMOD*. PGS98b/LSRACM, June 1996, pp. 173–182.

[7] "JBoss Application Server." [Online]. Available: http://www.jboss.org/jbossas/

[8] TPC Council, "TPC-C Benchmark." [Online]. Available: http://www.tpc.org/tpcc

[9] "Infinispan." [Online]. Available: http://www.jboss.org/infinispan

[10] "JCache (JSR-107)." [Online]. Available: http://jcp.org/en/jsr/detail?id=107

[11] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the world wide web," in *Proc. of STOC*, May 1997, pp. 654–663.

[12] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ansi sql isolation levels," *SIGMOD Rec.*, vol. 24, pp. 1–10, May 1995.

[13] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*, 1st ed. Morgan Kaufmann Publishers Inc., 1992.

[14] "JGroups." [Online]. Available: http://www.jgroups.org

[15] K. P. Birman and T. A. Joseph, "Reliable communication in the presence of failures," *ACM Trans. Comput. Syst.*, vol. 5, pp. 47–76, January 1987. [Online]. Available: http://doi.acm.org/10.1145/7351.7478

[16] "Ehcache." [Online]. Available: http://ehcache.org/documentation/overview.html

[17] "Oracle Coherence." [Online]. Available: http://coherence.oracle.com

[18] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, pp. 35–40, April 2010.

[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 205–220, October 2007.

[20] M. K. Aguilera, A. Merchant, M. Shah, A. Veitch, and C. Karamanolis, "Sinfonia: a new paradigm for building scalable distributed systems," *SIGOPS Oper. Syst. Rev.*, vol. 41, pp. 159–174, October 2007.

[21] D. R. K. Ports, A. T. Clements, I. Zhang, S. Madden, and B. Liskov, "Transactional consistency and automatic management in an application data cache," in *Proc. of OSDI*. USENIX Association, October 2010, pp. 1–15.

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

[23] B. Kemme and G. Alonso, "A suite of database replication protocols based on group communication primitives," in *Proc. of ICDCS*. IEEE CS, May 1998, p. 156.

[24] N. Schiper, R. Schmidt, and F. Pedone, "Optimistic Algorithms for Partial Database Replication," in *Proc. of OPODIS*. LNCS, Springer, December 2006, pp. 81–93.

[25] D. Serrano, M. Patino-Martinez, R. Jimenez-Peris, and B. Kemme, "Boosting Database Replication Scalability through Partial Replication and 1-Copy-Snapshot-Isolation," in *Proc. of PRDC*. IEEE CS, December 2007, pp. 290–297.

[26] N. Shavit and D. Touitou, "Software transactional memory," in *Proc. of PODC*. ACM, Aug 1995, pp. 204–213.

[27] C. Kotselidis, M. Ansari, K. Jarvis, M. Lujn, C. Kirkham, and I. Watson, "DiSTM: A software transactional memory framework for clusters," in *Proc. of ICPP*. IEEE CS, September 2008, pp. 51–58.

[28] R. L. Bocchino, V. S. Adve, and B. L. Chamberlain, "Software transactional memory for large scale clusters," in *Proc. of PPoPP*. ACM, February 2008, pp. 247–258.

[29] M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues, "D^{2}STM: Dependable distributed software transactional memory," in *Proc. of PRDC*. IEEE CS, November 2009, pp. 307–313.

[30] N. Carvalho, P. Romano, and L. Rodrigues, "Asynchronous Lease-Based Replication of Software Transactional Memory," in *Proc. of Middleware*. LNCS, Springer, November 2010.

[31] R. Palmieri, F. Quaglia, and P. Romano, "AGGRO: Boosting stm replication via aggressively optimistic transaction processing," *Proc. of NCA*, pp. 20–27, 2010.