Evaluating Database-oriented Replication Schemes in Software Transactional Memory Systems

Roberto Palmieri and Francesco Quaglia
DIS, Sapienza University, Rome, Italy

Paolo Romano and Nuno Carvalho
INESC-ID, Lisbon, Portugal

Abstract

Software Transactional Memories (STMs) are emerging as a highly attractive programming model, thanks to their ability to mask concurrency management issues to the overlying applications. In this paper we are interested in dependability of STM systems via replication. In particular we present an extensive simulation study aimed at assessing the efficiency of some recently proposed database-oriented replication schemes, when employed in the context of STM systems. Our results point out the limited efficiency and scalability of these schemes, highlighting the need for re-designing ad-hoc solutions well fitting the requirements of STM environments. Possible directions for the re-design process are also discussed and supported by some early quantitative data.

1 Introduction

Replication is a typical mean for achieving fault-tolerance and high availability. For transactional systems, state of the art replication solutions [7, 8, 11] are based on the integration of:

(i) An Atomic Broadcast (AB) service [3], ensuring replicas' agreement on a common Global Serialization Order (GSO) for the transactions.

(ii) A deterministic concurrency control scheme, guaranteeing that, at each replica, the schedule of the locally executed transactions yields a serialization order equivalent to the GSO defined via the AB service.

These solutions have been traditionally targeted to database systems. However, great attention has been recently devoted to Software Transactional Memories (STMs) [2], which leverage on the proven concept of atomic and isolated transaction to spare programmers from the pitfalls of conventional lock-based synchronization and to simplify the development of concurrent applications.

In this work we are interested in evaluating replication solutions originally tailored for database systems in the context of STM systems. The need for a study assessing the efficiency on these solutions in this novel transactional context is motivated by several factors.

First, transactions' execution times in (non-replicated) STM systems are typically orders of magnitude smaller than in conventional database environments [14], which leads to an amplification of the relative cost of the distributed replica coordination scheme. This can not only have significant negative effects on the transaction completion time, but could also cause under-utilization of the available computing resources (due to relatively longer stall periods while handling transaction processing activities) that could become particularly manifest in modern multi-core and massively parallel architectures.

A second factor is related to the different data layouts and conflict patterns among transactions, which, together with the specific concurrency control scheme regulating local processing activities, play a crucial role in determining the actual performance of AB-based replication techniques.

Actually, advanced replication techniques for database systems [8] rely on an optimistic variant of AB, normally referred to as Optimistic Atomic Broadcast (OAB). The OAB layer delivers any request to the transactional system in an optimistic fashion, with the meaning that it is still unknown the correct position of the corresponding transaction within the GSO. Successively, a final delivery occurs, notifying that position. The optimistic delivery phase allows early knowledge about the existence of the transactional request. In cases where spontaneous ordering properties hold (namely the optimistic delivery order well matches the GSO), early knowledge about request existence is used to immediately start transaction processing activities, so to overlap coordination and computing phases. This has been actuated in combination with a lock-based concurrency control scheme where each optimistically delivered transaction gets activated on the transactional system only in case it is deterministically ensured to run to completion.

The above scheme assumes the ability for the transaction to acquire all the locks it needs at startup, which will be released only upon transaction commit/rollback (i.e. upon GSO notifications by the OAB). Any other successively delivered and conflicting transaction is activated only after the preceding transaction releases its locks. The ability of this approach to effectively overlap communication and processing phases results therefore dependent on the trans-

action conflict patterns exhibited by the overlying applications.

Beyond efficiency dependency on the actual transaction conflict level, a relevant aspect for the viability of this approach is that it requires a-priori knowledge of both read and write sets associated with transactions, in order to a-priori identify conflicting transaction classes. As for this aspect, the difficulty to exactly identify the data items to be accessed by transactions before these are actually executed, may lead to the adoption of conservative conflict assumptions based on coarse data granularity, e.g. whole, or large slices of, database tables [13]. However, unlike relational database systems, STM-based applications are characterized by arbitrary memory layouts and access patterns, which make harder, or even impossible, to a-priori identify, with a reasonable accuracy (or even a reasonable over-estimation), the boundaries of the memory regions that will be accessed by transactions prior to their execution. On the other hand, large over-estimation of the actual transaction conflicts is an additional performance adverse factor since it can strongly hamper concurrency, leading to significant resources' under-utilization in (massively) parallel systems.

The aforementioned phenomena are analyzed in this paper via an extensive simulation study based on data access traces of commonly used (S)TM benchmark applications [6]. The target replication protocol is exactly the one presented in [8]. As hinted, this is a database-oriented optimized solution relying on the OAB variant of total order group communication primitives. Our study shows that, when employed in STM contexts, this protocol is prone to significant CPU under-utilization in modern multi-core architectures. We show how this phenomenon is strictly related to the significantly higher conflict rate among transactions exhibited by the STM benchmarks, compared to benchmarks for database systems [15]. Also, the performance benefits (in terms of execution latency reduction) achievable by overlapping communication and processing activities is extremely limited, especially for finer grain transactions. Successively, we show how even minor modifications of the transaction management logics (e.g. in terms of concurrency control) can lead to significant reduction of transactions' latency, motivating the need for revisiting/re-designing replication solutions explicitly optimized for STM systems. Promising directions for such a revision/re-design process are also discussed.

The remainder of this paper is structured as follows. In Section 2 we describe the architecture of the replicated STM system under investigation. The simulation model, the benchmark applications and the performance results are provided in Section 3. Section 4 concludes the paper.

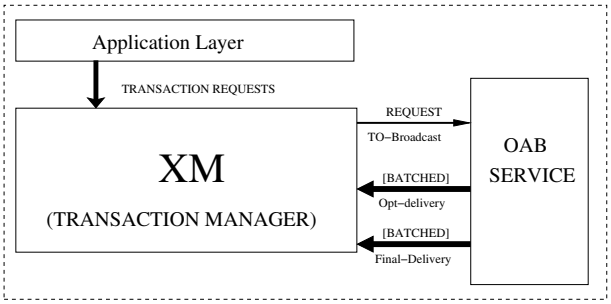

Figure 1. The Architecture of a Replicated STMP.

2 System Architecture

Our performance study is targeted to a typical system architecture consisting of a set of replicated Software Transactional Memory Processes (STMPs) relying on an OAB service offering the following classical API:

- *TO-broadcast*(m), which allows broadcasting messages to all the replicated STMPs.
- *Opt-deliver*(m), which delivers the message m to an STMP in a tentative, also called optimistic, order.
- *TO-deliver*(m), which delivers the message m to an STMP in a so called *final order*, which is the same for all the replicas.

With no loss of generality, we assume that upon the invocation of *TO-deliver*(m), the message m is exactly the next one finally-ordered by the OAB service, as is the case for most implementations (see, e.g., [12]).

We consider both the case of Opt/TO delivered messages carrying a single transactional request, and the scenario in which the OAB service employs batching mechanisms. In the latter case, each Opt-delivered message carries a batch of transactional request messages, for which the corresponding TO-delivery defines the total order with respect to other (batched) requests. Clearly, each batch contains messages ordered according to, e.g., the corresponding position within the batch. We recall that batching is a technique very commonly employed to optimize the performance of (Optimistic) Atomic Broadcast protocols [3]. By amortizing the costs associated with the (O)AB execution across a set of messages, batching schemes have been shown to yield considerable enhancement of the maximum throughput achievable by (O)AB protocols. The inclusion of batching schemes in our study of OAB-based replication protocols for transactional systems allows keeping into account optimized configurations for this important building-block group communication primitive.

The diagram in Figure 1 shows the architecture of each STMP. Applications generate transactions by issuing requests to the local Transaction Manager (XM), specifying the business logic to be executed (e.g. the name of a method/function of a transactional memory-based application) and the corresponding input parameters (if any). XM is responsible of (i) propagating/collecting transactional requests through the OAB service, (ii) executing the transactional logic, and (iii) returning the corresponding result to the user-level application.

We assume that XM regulates concurrency among transactions by exactly employing the scheme proposed in [8]. As hinted, this scheme aims at an overlap between coordination and computing phases. In particular, upon its optimistic delivery, a transaction gets immediately activated in case it is known not to conflict with any active transaction. In the negative case, the transaction is simply queued, and gets activated only upon commit/rollback of the currently active conflicting transaction(s). Transaction rollback occurs in case the OAB service notifies that conflicts have not been serialized according to the GSO. In such a case, the rolled-back conflicting transactions are then re-executed (in a sequential mode) and serialized according to the GSO. This type of concurrency control actually represents a variant of locking where:

- For each transaction, there is an a-priori knowledge (or conservative estimation) of the set of required locks, and hence of the transaction read/write set.
- Upon startup, a transaction needs to acquire all its required locks.
- Locks are released all together upon transaction commit/rollback.

In [8] it has been discussed how to simulate this type of locking scheme on top of conventional database systems. This has been done via a stub that performs pre-analysis of transactions statements in order to determine the data to be accessed, and by maintaining a so called lock-table indicating whether the requested data are not currently locked (in which case transaction execution can be started), or, in case they are locked, the identity of both active and already queued transactions requesting access to those data (in which case the transaction gets queued within the lock-table). Transaction queues within the lock-table are reorganized in case TO-deliveries subvert the corresponding Opt-deliveries, which originally determined the order of transactions' enqueue.

3 Simulation Study

In this section we report the results of an extensive simulation study aimed at assessing the performance of the OAB-based transactional replication scheme proposed in [8] when employed to enhance the dependability (in terms of fault-tolerance and high availability) of applications based on replicated STM systems structured as depicted in the previous section.

We start by describing the simulation model and the employed benchmark applications. Then we present the results of the performance analysis, shedding light on some significant limitations of the approach in [8] when adopted in STM environments. Finally, we provide assessments on possible approaches for tackling those limitations, by also reporting quantitative data related to the performance improvements that could be achieved by adopting some modifications over [8] for the operating mode of the replicated STM system.

3.1 Simulation Model

Our performance evaluation study is based on a process-oriented simulator developed using the JavaSim simulation package [10]. As said, the XM layer at each replicated process collects transactional requests via the OAB layer before activating any processing activity. The transactions' arrival process via Opt/TO message deliveries from the OAB layer is modeled in our simulations via a message source that injects messages having as payload a batch of β transactions with an exponentially distributed inter-arrival rate, having mean λ. In order to derive results representative of a wide range of settings, we let the batching size β vary in between 1 and 8.

In order to accurately model the execution dynamics of transactions in STM systems, we rely on a trace-based approach. Traces related to data accesses and transaction duration have been collected by running a set of widely used, standard benchmark applications for (S)TMs. The machine used for the tracing process is equipped with an Intel Core 2 Duo 2.53 GHz processor and 4GB of RAM. The operating system running on this machine is Mac OSX 10.6.2, and the used STM layer is JVSTM [1]. The simulation model of the replicated STM system comprises a set of 4 replicated STMPs, each of which hosted by a multi-core machine with 8 cores exhibiting the same power as in the above architecture.

We configured the benchmarks to run in single threaded mode, so to filter out any potential conflict for both hardware resources and data. Also, we extended JVSTM in order to transparently assign a unique identifier to every object within the STM memory layout and to log every operation (namely, begin/commit/rollback operations, and read/write memory-object access operations) along with its timestamp. This allowed us to gather accurate information on the data access patterns of the benchmark applications and on the time required for processing each transaction (in absence of any form of contention).

Since any tracing strategy unavoidably introduces overheads, especially in STM applications where transaction ex-

ecution times are often less than 1 msec, in order to ensure the accuracy of the information concerning the duration of transactions, we repeated each benchmark run (ensuring the deterministic re-execution of an identical set of transactions) by also disabling the logging functionality. Then we compared the resulting mean transaction execution time with the one obtained when logging is enabled. This allowed us to compute a per-benchmark scaling factor (that, on the average, was found to be around 15x) used to adjust the duration of the transaction execution, thus filtering out the overheads associated with the logging layer, before feeding this information to the simulator.

The traces were collected running three benchmark applications, RB-Tree, SkipList and List, that were originally used for evaluating DSTM2 [6] and, later on, adopted in a number of performance evaluation studies of (S)TM systems [1, 2]. These applications perform repeated insertion, removal and search operations of a randomly chosen integer in a set of integers. The set of integers is implemented either as a sorted single-linked list, a skip list, or a red-black tree. We configured the benchmark to initialize the set of integers with 128 values, and allowed it to store up to a maximum of 256 values. Finally, we configured the benchmark not to generate any read-only transaction (i.e. searches). This has been done in compliance with the operating mode of the protocol in [8], where read-only transactions can be executed locally at each single replicated process, without the need for propagation via the atomic broadcast (read-only transactions do not alter the state of the replicated transactional memory system). By only considering update transactions in our study, we can therefore precisely assess the impact of the atomic broadcast latency on the performance of a replicated STM, as well as the performance gains achievable by means of the optimistic approach proposed in [8]. The below table reports the average transaction execution time observed for the three benchmarks via the aforementioned tracing scheme:

Benchmark	Avg. Transaction Exec. Time
RB-Tree	77 μsec
SkipList	281 μsec
List	324 μsec

As for the delay due to the OAB layer (i.e. the delay of the Opt-delivery and of the corresponding TO-delivery), several studies have shown that OAB implementations typically tend to exhibit flat message delivery latency up to saturation [4]. On the other hand, our study is not targeted to explicitly assess the saturation point of the OAB group communication subsystem. For this reason we decided to run the simulations by assuming that the OAB layer does not reach its saturation point. Therefore, independently of the value of the message arrival rate λ, we use in our simulations an average latency of 500 microseconds for the Opt-delivery, and of 2 milliseconds for the TO-delivery. These values have been selected based on experimental measures obtained running the Appia [9] GCS Toolkit on a cluster of 4 quad core machines (2.40GHz - 8GB RAM) connected via a switched Gigabit Ethernet.

Finally, we consider an ideal scenario with no mismatch between the optimistic and the final message delivery order. This is because the protocol in [8] has been explicitly targeted to environments where such a spontaneous ordering property holds (e.g. LAN based environments). This clearly represents a best-case scenario for the protocol in [8], which allows us to establish an upper bound on the performance achievable by this protocol vs the behavior of the OAB layer (in terms of actual message ordering).

3.2 Simulation Results

In Figure 2 we report response time and CPU utilization values for the three benchmarks, while varying the arrival rate of transactional requests λ from the OAB layer. We plot results related to 4 different configurations for the replication protocol. Opt-Fine and Opt-Coarse refer to the values observed when employing the replication protocol in [8] by using either the actual transaction conflicts (determined by using the exact accesses to memory objects as logged within the benchmark execution traces) or a coarse conservative estimation where each pair of concurrent transactions is assumed to always conflict on some object within the STM memory layout. For these two different conflict patters we also report the performance observed via a State Machine (SM) approach relying on traditional, non-optimistic Atomic-Broadcast, where transactions are activated only after the group communication layer has determined their correct position within the GSO.

By the results we can observe two major tendencies. For all the benchmarks, CPU utilization at the saturation point is always lower than 20%. Given that in our simulation the OAB layer is configured to respond in its flat region, this is a clear indication that data conflicts are the cause of system saturation. The worst case for the impact of data conflicts on the saturation point can be observed for List, where the CPU utilization does not even reach 6%, and the response time curves in case of actual conflicts exactly coincide with the corresponding ones obtained via coarse conflict estimations. The latter phenomenon is less evident for the other benchmarks, especially RB-Tree. However, it is a clear indication that data access patterns in STM environments may anyway exhibit actual conflict levels significantly grater than in database applications. This motivates the need for investigating optimized concurrency control schemes, especially when employed in replicated environments where transaction blocking up to the determination of the final GSO for already active conflicting transactions may have a strong negative impact on resources' under-utilization.

The second tendency we can observe is related to limited advantages, in terms of transaction execution latency, from the overlap between coordination and computing phases in the Opt scheme compared to SM. This is noted especially for the very fine grain RB-Tree benchmark, exhibiting mean transaction execution time significantly less than the delay of TO-deliveries. Such a reduced transaction granularity, in combination with non-minimal transaction conflict levels (observed even for the fine conflict determination approach based on actual accesses), significantly reduces the performance gains achievable with respect to non-optimistic approaches. This depends on the fact that the coordination phase overlaps with a very reduced amount of fine grain computing activities, whose individual delay is actually negligible compared to the coordination latency.

As a final observation, results with batching factor β greater than 1 are not reported since they almost coincide with those observed with $\beta = 1$. This indicates that, in STM environments, the replication management scheme in [8] does not take advantages (in terms of performance of transaction processing) when employed in combination with optimized group communication modules relying on batching mechanisms.

3.3 Assessments

Based on the results in the previous section, we argue that replication schemes originally designed for databases need to be significantly revised in order to deliver adequate performance in STM contexts. We envisage at least two ways according to which the revision could be actuated:

(A) The concurrency control scheme supported by XM should increase the overlap level between coordination and computing phases. This would entail increasing the level of optimism in transaction processing activities by avoiding, e.g., the block-until-GSO phase in case of Opt-delivered transactions that conflict with already active ones (for which GSO determination is on-going). Such an aggressive-optimistic approach can provide advantages in terms of response time reduction, with the maximum benefits being expectable in those scenarios where the Opt-delivery order is likely to match the final TO-delivery. This is because optimistic anticipation of the execution of chains of conflicting transactions, before the corresponding GSO gets determined, will not likely result in cascading rollback scenarios, otherwise caused by discrepancies between Opt-delivery and TO-delivery orders when spontaneous ordering properties do not hold. This approach should anyway provide no advantage in terms of CPU utilization for processing activities associated with spontaneously ordered optimistically delivered transactions since, compared to the scheme in [8], an increased level of optimism (combined with spontaneous ordering) would simply yield to anticipate the execution of conflicting transactions before the GSO is actually defined for the oldest transaction along the conflict-serialized order.

(B) Amortizing response time penalties that could arise when spontaneous ordering is not guaranteed by concurrently exploring differentiated serialization orders in a speculative fashion, according to the aggressive-optimistic scheme provided in the above point. This would lead to increasing hardware resources' utilization without negatively impacting response time. The set of serialization orders to be explored while notification of the correct one (i.e. the GSO) takes place via the finalization of the OAB service could be determined using heuristics or a clear theoretical analysis, allowing the avoidance of redundant execution of equivalent serialization orders on the basis of actual transaction conflicts.

As for point (A), we have early results related to transaction response time improvements, which can be achieved in case of a basic aggressive-optimistic scheme in which an Opt-delivered transaction T gets activated as soon as all the other Opt-delivered transactions preceding T in the optimistic delivery order have completed their execution (although they might be still unresolved in terms of commit/rollback due to ongoing determination of the corresponding GSO). Such a scheme would entail controlling concurrency in a way that each transaction T, serialized after T' according to the optimistic delivery order, is allowed to access to the post-image of memory objects with respect to the execution of T'. Similar approaches have been studied in literature in the context of Distributed Atomic Commit schemes (see [5]) for affording temporal unavailability of precommitted data by allowing (at least at a limited degree) the exposition of data post-images to a subsequent transaction.

The percentage response-time reduction via the above aggressive-optimistic scheme over the original proposal in [8] is reported in Figure 3 for all the three benchmark applications already employed in the first part of this study. By the results we observe that, as soon as the transactional request arrival rate gets increased, the aggressive-optimistic approach provides significant reductions of the response time up to saturation (still reached due to data conflicts). The peak performance gain leads to a reduction of up to 35% in the response time. Also, the reduction is observable for wider workload intervals when the batching factor β at the OAB layer gets increased. In particular, batch level $\beta = 8$ allows 35% response time reduction for both Skip-list and List even at low message arrival rates, thus showing how an aggressive-optimistic approach has the capability to tackle bursts of transactional requests (carried by

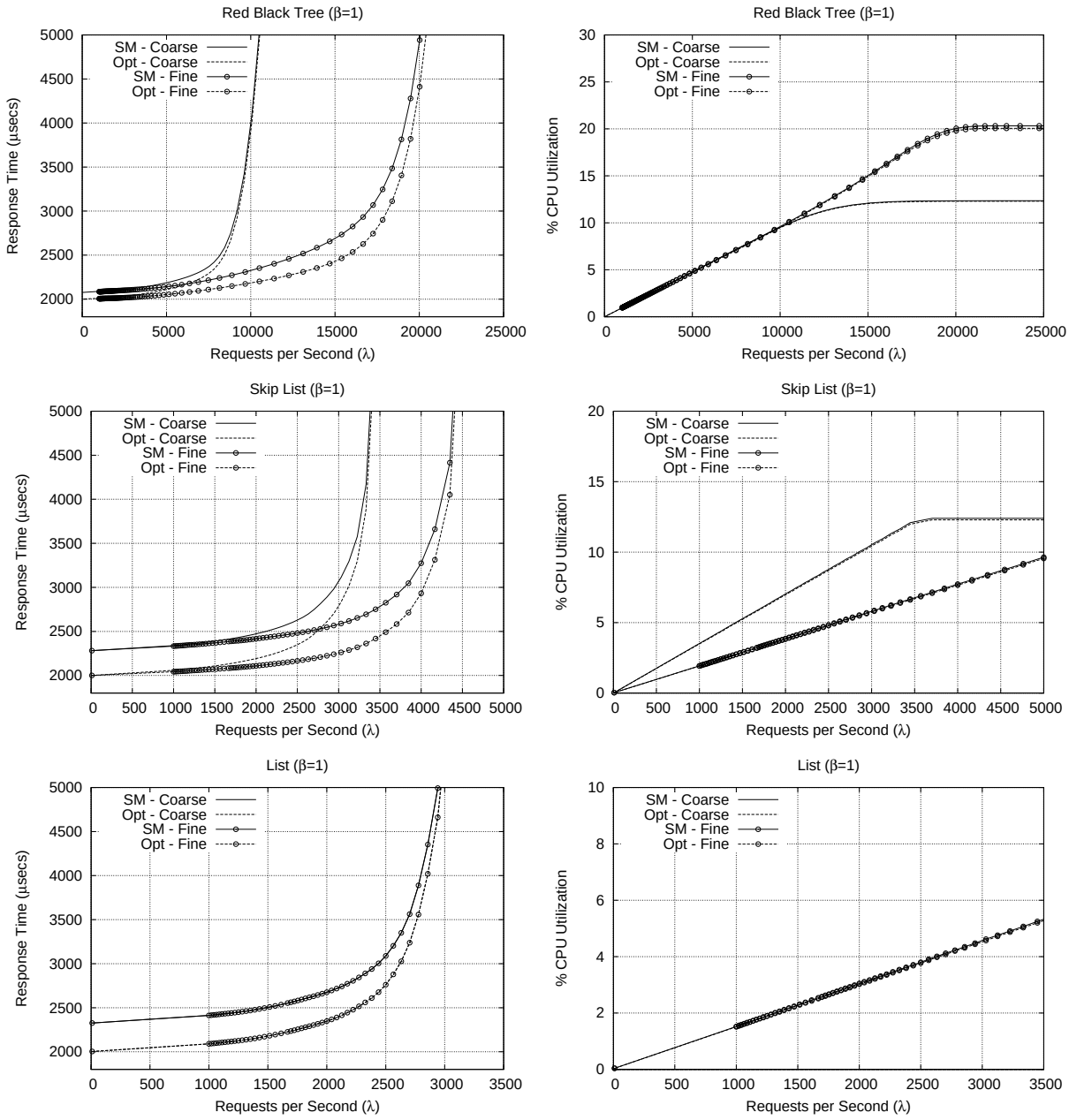

Figure 2. Performance Results for SM and Opt.

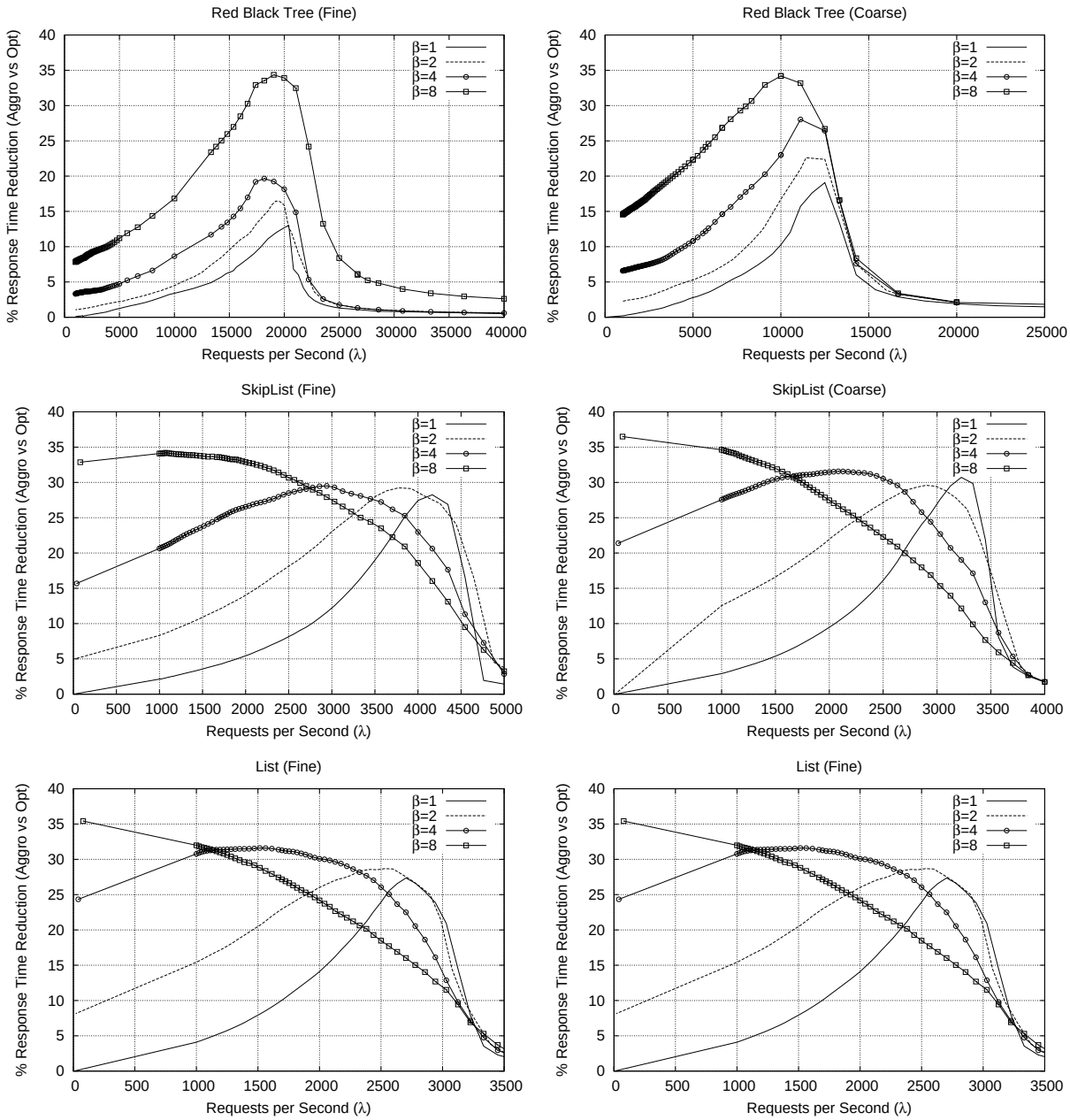

Figure 3. Performance Benefits by the Aggressive-Optimistic Approach.

the batched Opt-delivered message) just thanks to the increased concurrency level, which in turn leads to increased actual overlap between coordination and communication. This phenomenon is less evident for the very fine grain RB-Tree benchmark. However, in case of non-minimal batching values and increased message arrival rate, such an increased overlap definitely favor response time also for RB-Tree.

4 Conclusions

In this work we have evaluated via simulation the usage of common database-oriented techniques for dependability, based on group communication primitives for replica coordination, in Software Transactional Memory systems. The first part of the simulation provides two main outcomes. First, both the traditional State Machine approach and the more recent approach based on optimistic atomic broadcast primitives lead to under-utilization of hardware resources (CPUs). Second, the approach based on optimistic atomic broadcast does not take significant advantages from the optimistic overlap between coordination and computing phases. In the second part of the simulation study, we have presented a relatively simple variant of the optimistic atomic broadcast based protocol, that aims at increasing the actual level of overlap between communication and processing. The results highlight that this approach can yield up to 35% reduction of the transaction execution latency. We have also discussed research lines for further improvements, e.g., in the direction of speculative processing along differentiated serialization orders.

Acknowledgments

This work was partially supported by the ARISTOS (PTDC/EIA-EIA/102496/2008) project and by FCT (INESC-ID multiannual funding) through the PIDDAC Program funds.

References

[1] J. Cachopo and A. Rito-Silva. Versioned boxes as the basis for memory transactions. *Sci. Comput. Program.*, 63(2):172–185, 2006.

[2] M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues. D^{2}STM: Dependable Distributed Software Transactional Memory. In *Proc. International Symposium on Dependable Computing (PRDC)*. IEEE Computer Society Press, 2009.

[3] X. Defago, A. Schiper, and P. Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, 2004.

[4] R. Ekwall and A. Schiper. Modeling and validating the performance of atomic broadcast algorithms in high-latency networks. In *Proc. Euro-Par 2007, Parallel Processing*, Lecture Notes in Computer Science, pages 574–586. Springer, 2007.

[5] J. R. Haritsa, K. Ramamritham, and R. Gupta. The PROMPT real-time commit protocol. *IEEE Transactions on Parallel and Distributed Systems*, 11:160–181, 2000.

[6] M. Herlihy, V. Luchangco, and M. Moir. A flexible framework for implementing software transactional memory. *SIGPLAN Not.*, 41(10):253–262, 2006.

[7] B. Kemme and G. Alonso. A suite of database replication protocols based on group communication primitives. In *Proc. of the 18th International Conference on Distributed Computing Systems*, page 156, Amsterdam, Netherlands, 1998. IEEE Computer Society.

[8] B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing transactions over optimistic atomic broadcast protocols. In *Proc. of the 19th IEEE International Conference on Distributed Computing Systems*, page 424, Austin, TX, USA, 1999. IEEE Computer Society.

[9] H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proc. 21st IEEE International Conference on Distributed Computing Systems*, pages 707–710, Phoenix, Arizona, Apr. 2001. IEEE.

[10] U. of Newcastle Upon Tyne. JavaSim 0.3 GA. http://javasim.codehaus.org/, 2009.

[11] F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Distributed and Parallel Databases*, 14(1):71–98, 2003.

[12] F. Pedone and A. Schiper. Optimistic atomic broadcast: a pragmatic viewpoint. *Theor. Comput. Sci.*, 291(1):79–101, 2003.

[13] F. Perez-Sorrosal, M. Pati no-Martinez, R. Jimenez-Peris, and B. Kemme. Consistent and scalable cache replication for multi-tier J2EE applications. In *Middleware '07: Proceedings of the International Conference on Middleware*, pages 328–347, New York, NY, USA, 2007. Springer-Verlag New York, Inc.

[14] P. Romano, N. Carvalho, and L. Rodrigues. Towards distributed software transactional memory systems. In *Proceedings of the Workshop on Large-Scale Distributed Systems and Middleware (LADIS 2008)*, Watson Research Labs, Yorktown Heights (NY), USA, ACM Press, Sept. 2008.

[15] Transaction Processing Performance Council. *TPC Benchmark^{TM} W, Standard Specification, Version 1.8.* Transaction Processing Perfomance Council, 2002.