Towards the Integration of Distributed Transactional Memories in Application Servers' Clusters *

Paolo Romano, Nuno Carvalho, Maria Couceiro,
Luís Rodrigues and João Cachopo

INESC-ID, Lisbon, Portugal

Abstract. The transition to multicore architectures has raised the urge to identify novel programming paradigms aimed at simplifying the development of parallel programs.
Transactional Memories (TM) are regarded as one of the most promising approaches to address this issue, as highlighted by the huge interest garnered in the research community over the last years. Distributed Transactional Memories (DTMs) represent a very recent branching of the research line on TMs, aimed at enhancing their scalability and dependability.
In this paper, we review some of our recent results and research directions focused on the integration of DTMs in clusters of web application servers and on the design of scalable and fault-tolerant DTM algorithms.

1 Introduction

Transactional Memories (TMs) have garnered considerable interest of late due to the recent technological trend that has made of multi-core and many-core CPUs the architecture-of-choice for mainstream computing. TMs represent an attractive solution to spare programmers from the pitfalls of conventional explicit lock-based thread synchronization, relying instead on proven concurrency-control concepts used for decades by the database community to simplify concurrent programming [1]. When using TMs, the programmers are simply required to specify which operations on shared data structures are to be executed within the scope of an atomic and isolated transaction. By relinquishing the programmer from the burden of managing locks or other error-prone low-level concurrency control mechanisms, TMs have been shown to enable a sensible boost in productivity, as well as in code reliability, e.g., [6].

Even though the study of TMs has garnered a large interest in the research community over the last 5 years, the problem of how to enhance their scalability and fault-tolerance via distribution and replication has started to receive attention only very recently [23, 4, 2, 12]. This is actually a major gap, which becomes pretty manifest when TMs start to be adopted in real world applications, as they

* This paper was partially supported by the Pastramy (PTDC/EIA/72405/2006) projects.

are faced with harsh scalability and dependability challenges that cannot be effectively tackled, due to the current lack of efficient Distributed Transactional Memory (DTM) solutions.

Web applications represent an important class of the systems that would significantly benefit from the adoption of TM-based solutions, provided that these are able to ensure adequate levels of scalability and failure resilience. Modern Web-based applications, in fact, tend to be structured according to a three-tier (or, more in general, multi-tier) architecture that relies on relational DBMSs for persisting the application data, whilst exploiting object-oriented programming platforms (e.g., J2EE) hosted by dedicated application servers for implementing the business logic. This allows reflecting at both the software and hardware level the logical decomposition of applications, permitting to achieve high modularity and flexibility. On the other hand, the partitioning of the application into multiple tiers generates an obvious increase in the system's complexity, generating performance and reliability pitfalls and hindering developers' productivity. Accessing the data on remote DBMSs, in fact, imposes incurring into onerous round-trips that may significantly hamper performance, especially for the case of complex operations. Further, the multiplicity and diversity of the employed components, and their interdependencies, makes reliability a complex issue to tackle, exposing the system to a spectrum of hazardous state inconsistencies in the presence of failures [14, 32]. Finally, rather than being a completely transparent aspect, relational-based persistence affects the programming model, hindering the successful implementation of an object-oriented rich domain model.

To overcome these problems, in [6, 9], we introduced a novel, TM-centric approach to architect web applications. In such an approach, the application's state is hosted in memory by the application servers, and locally persisted for scalability and durability purposes. The increased locality between the application logic and data alleviates the aforementioned performance and reliability issues. Accesses to the application state are handled, in a totally transparent manner for the developers, by a DTM layer that (i) enforces the atomicity and the isolation of any state updates triggered by the application, (ii) guarantees the consistency of the application state replicated across the nodes of the cluster, and (iii) triggers, when necessary, the update of a lightweight persistent storage system that is also replicated across the cluster.

In this paper we describe some of our recent results towards the realization of the envisioned TM-centric architecture, focusing in particular on the issues related to the design and implementation of scalable and dependable DTMs (rather than, e.g., on the aspects related to persistence).

We start by reporting our experiences with the development of the Fénix-EDU application, which represents, to the best of our knowledge, the first web application to have leveraged on TM technology. Next, we report the results of a workload characterization study of FénixEDU, whose results have driven some of our main choices in the design space of the algorithms architected to ensure the consistency of DTM platforms. We then describe BFC (Bloom Filter Certification), a novel TM replication protocol, which we recently proposed in [12], that exploits an efficient Bloom Filter-based encoding technique to reduce the

overhead associated with the cluster-wide certification of transactions. Finally, we point out some of our current research directions in this area.

This paper is organized as follows. Section 2 presents related work. Section 3 introduces the FénixEDU system, describing its current architecture and highlighting some key aspects of its workload. In Section 4 we illustrate the proposed architecture. The BFC replication protocol is overviewed in Section 5. Section 6 concludes the work and points to future research directions.

2 Related work

One way to implement a web application domain model is to use an object-oriented paradigm. A common approach is, for instance, to use a multi-tier J2EE architecture where the business logic and data are modeled using Enterprise Java Beans, being the data stored in databases by means of an Object/Relational mapping tool. The solution proposed in [29] presents a way to replicate such systems by adding fault tolerance mechanisms on both the application server and the database. The authors rely on a locking based approach. In our case, by relying on a DTM to synchronize concurrent accesses directly at the application server's tier, we are able to avoid error-prone, explicit locking schemes, and to rely on much simpler, and more lightweight, persistence solutions rather than on fully-fledged relational databases.

The only DTM solutions that we are aware of are those in [23, 4, 2]. However, the solutions proposed so far have not addressed the important issue of how to exploit replication not only to improve performance, but also to enhance dependability. This is clearly a central aspect of DTM's design, as the probability of failures increases with the number of nodes, becoming impossible to ignore in large clusters.

The problem of replicating a TM is closely related to the problem of database replication, given that both TMs and DBMSs share the same key abstraction of atomic transactions. The fulcrum of modern database replication schemes [28, 27] is the reliance on an Atomic Broadcast (ABcast) primitive [13, 19], typically provided by some Group Communication System (GCS) [25]. Roughly speaking, an ABcast service ensures that the broadcast messages are received by all or none of the participants, and in the same order, despite the occurrence of failures. ABcast plays a key role to enforce, in a non-blocking manner, a global transaction serialization order without incurring in the scalability problems affecting classical eager replication mechanisms based on distributed locking and atomic commit protocols, which require much finer grained coordination and fall prey of deadlocks [16]. Certification-based approaches, such as [28, 30, 29] are considered as some of the most efficient solutions among the plethora of ABcast based replication schemes [12]. Therefore, they represent natural candidates also in the context of TM systems. In these schemes, transactions are locally processed on a single replica and validated *a posteriori* of their execution through an ABcast based certification procedure aimed at detecting remote conflicts between concurrent transactions. The certification based approaches may be further classified

into voting and non-voting schemes, where voting schemes, unlike non-voting ones, need to ABcast only the writeset (and not the readset which may be very large), but on the other hand incur in the overhead of an additional uniform broadcast [19] along the critical path of the commit phase. As highlighted in our previous work [31], the replica coordination latency has a significantly amplified cost in TMs when compared to conventional database environments, given that the average transaction execution time in TM settings is typically several orders of magnitude shorter than in database ones. To maximize efficiency, it is therefore highly desirable to design novel mechanisms capable of minimizing the costs associated with the replica coordination schemes. This represents an important goal of our current research activities.

Our work is also related to the large body of literature on Distributed Shared Memories (DSM). To overcome the strong performance overheads introduced by classic DSM implementations [35, 24], which ensure strong consistency guarantees with the granularity of a single memory access, several DSM systems provide relaxed memory consistency guarantees, e.g., [21]. Unfortunately, developing software for relaxed DSM's consistency models may be challenging for programmers because they need to master complicated consistency properties. Conversely, the simplicity of the atomic transaction abstraction, at the core of (D)TMs, allows to increase programmers' productivity [6] with respect to both locking disciplines and relaxed memory consistency models. Further, the strong consistency guarantees provided by atomic transactions can be supported through efficient algorithms that incur only in a single synchronization phase per transaction (typically taking place at commit time), effectively amortizing the communication overheads across a set of (possibly large) memory accesses.

3 The FénixEDU system

The FénixEDU system is a web application that supports a wide range of academic activities in the Lisbon's Instituto Superior Técnico (IST) campus (management of web pages for different courses, student enrollment, etc). The FénixEDU system started as a typical web application, with the application logic implemented in Java and hosted by a single application server, and its data stored in a relational DBMS. In its first version, FénixEDU relied on an Object/Relational mapping tool [26] to store the objects in the database, while maintaining a local cache of the database data. To control the concurrent access to the domain entities, FénixEDU relied on explicit lock-based interfaces to synchronize read and write operations [11]. Unfortunately, this lock-based approach to concurrency was highly error-prone, as programmers often forgot or misplaced the acquisition of locks, causing frequent consistency problems into the domain data. Moreover, with the increased usage of the system, also the first performance problems appeared. After some performance profiling, these were attributed to the overheads incurred in the acquisition and in the management of locks by the operations that accessed many thousands of objects.

To address these issues, across 2005 the FénixEDU codebase was adapted to permit transparent integration with a TM layer, called JVSTM [7]. JVSTM

relies on a software based implementation of a multiversion concurrency control scheme [3], providing excellent performance for read-only transactions (largely predominating in reference benchmarks for Web applications [36, 37], as well as in the FénixEDU's workload), because they incur in negligible book-keeping overheads and are sheltered from the possibility of blocks or aborts. The integration of JVSTM within the architecture of the FénixEDU application was designed so to achieve total transparency from the developer's perspective, and provided benefits not only in terms of performance (thanks to the elimination of the overheads associated with lock acquisition and management), but also in terms of robustness (thanks to the avoidance of the error-prone manual management of locks) and simplification of the programming model (quantifiable in terms of reduction of lines of code to be developed and debugged [6]).

Serving a population of 12000 students, 900 faculty and 800 administrative members and faced with a steadily increasing traffic volume, the FénixEDU system was eventually forced to address the problem of scaling out the TM-enabled application server. As a first step in this direction, a very simple replica synchronization scheme orchestrated by a centralized back-end database is currently being employed. Essentially, each application server is required to access the database every time it starts a new transaction (whether read-only or not) to check whether its local cache is still up-to-date. The detection of any conflict developed during transactions' execution is performed at commit time via a sequential validation phase performed by checking whether the readset of the committing transaction T intersects with the writesets (stored by the database) of any transaction that has committed before T. Unfortunately, even though this approach is very simple, the reliance of the current replication solution on an external data storage causes large performance overheads, strongly limits concurrency, and suffers of a single point of failure.

To drive the design phase of an efficient DTM platform capable of effectively matching the characteristics of the FénixEDU application, the system was instrumented to collect information on the nature of the workload generated by the application towards the TM layer. The workload data was collected over a period of two years (from June 2007 to July 2009), gathering information concerning around 390 millions of transactions. A first result that we observed is that write transactions are approximately only 2% of the total number of transactions in the system. This ratio supports the choice of a TM layer, such as JVSTM, particularly optimized for read-only transactions. Further, it suggests to bias the design of a DTM platform so to require synchronization exclusively for write transactions, unlike the currently operational solution that demands a remote synchronization with the back-end also when a read-only transaction is started.

In Figure 1 we plot the probability density functions of the readset's and writeset's sizes for write transactions, as these factors may have a big influence on the amount of information exchanged among the replicas of a DTM. From these plots we can draw two main considerations. On one side, we observe that, on average, writesets are several orders of magnitude smaller than readsets (more precisely the average readset's size is of 5575, whereas the average writeset's size

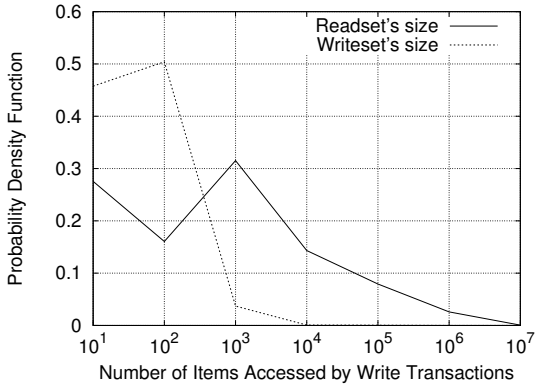


Fig. 1. Probability density functions for the readset's and writeset's size of write transactions.

is of 36). This suggests that DTM solutions should strive to avoid communicating the whole readset and, whether possible, should exclusively propagate information concerning the transactions' writeset. On the other hand, the sizes of the readset and the writeset are far from being concentrated around their average values: the maximum readset's and writeset's sizes, in fact, are from three to four orders of magnitude larger than the corresponding average values. In other words, the FénixEDU's workload comprehends very heterogeneous components, which makes it extremely challenging to identify a "one-size-fits-all-solution" capable to deliver optimal performances in every scenario.

4 System Architecture

To address the above discussed inefficiencies and limitations affecting the DTM solution currently supporting the FénixEDU application, in [9] we have recently proposed a new architecture that neatly decouples the issues related with the synchronization of the replicated TM layer with those concerning the persistence of data. In the envisioned architecture, which is depicted also in Figure 2, the consistency of the TM-enabled application server replicas is no longer dependent on a centralized DBMS. Conversely, the application server replicas coordinate the execution of transactions by directly communicating among them, leveraging on the services provided by a Group Communication System to propagate the updates and reach cluster-wide agreement on the outcome (commit/abort) of transactions.

Coping with the issues related to the concurrent execution of distributed transactions directly at the application server replicas' level, rather than relying on the assistance of a standard relational DBMS, provides two main advantages.

First, the achievement of a neater separation of logical concerns: once the TM tier has autonomously ensured the consistency of a transaction, the back-end just has to be able to atomically persist the corresponding updates. This

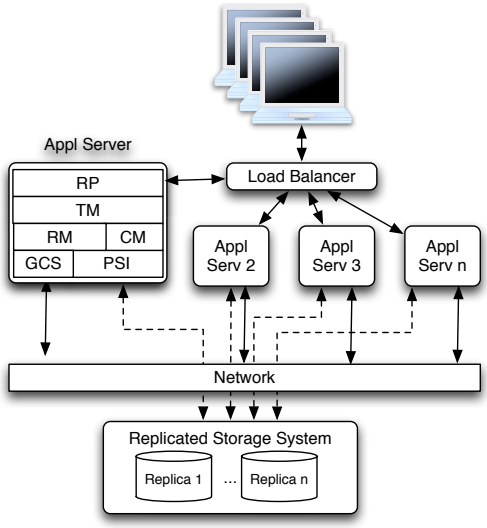


Fig. 2. The proposed DTM based architecture.

permits to rely on much simpler, and more lightweight, persistence solutions than fully-fledged relational databases (which incur in the unnecessary overheads associated with SQL or complex concurrency control mechanisms [34]).

Further, the reliance on a standard, relational DBMS as the coordinator of the TM replication protocol forces to implement the whole replication protocol by exploiting exclusively standard SQL interface. Being SQL designed for other purposes, it can significantly hinder the development of even basic mechanisms typically employed by any transactional replication scheme (e.g., synchronous propagation of state updates to other replicas).

Let us now analyze more in detail the architecture illustrated in Figure 2. The incoming requests are dispatched by a load-balancer (see [8] for a comprehensive survey on load-balancing in web clusters) to a set of replicated application servers, which rely on a replicated persistent storage for ensuring durability. Note that the latter is depicted as a logical independent component, even though it could be physically colocated in the same machines also hosting the application server to enhance locality between the application logic and the (persistent) data. In the remainder, we will concentrate on the description of the modules composing the DTM layer, which represents the actual focus of this paper, postponing a thorough analysis of the replicated persistence storage to a future paper.

Each application server hosts the following components: a *Request Processor* (RP), responsible for receiving the requests and activating the transactional logic; a *TM* instance, extended with a reflective interface that externalizes key information about the transactions' execution state (e.g., transactions' readsets and writesets), which are normally encapsulated by existing TM implementations; a *Cache Manager* (CM), responsible for implementing caching policies (e.g., prefetching, eviction strategies) based on the application access patterns;

a *Replication Manager* (RM), implementing the distributed coordination protocol required for ensuring replica consistency (an overview of the TM replication protocols currently under development/evaluation will be provided in the remainder of this paper); a *Persistent Store Interface* (PSI), providing APIs to interact with a replicated storage system; a *Group Communication Service* (GCS), responsible for maintaining up-to-date information regarding the membership of the group of application servers (including failure detection) and providing the required communication support for the coordination among the servers.

5 The BFC Protocol

The Replication Manager, being in charge of ensuring the consistency of the DTM layer, represents a fundamental building block of the architecture described in Section 4. In this section we present one of our recent results concerning the design and evaluation of TM replication mechanisms, namely the Bloom Filter Certification (BFC) protocol [12].

As already discussed in Section 2, the abundant literature on database replication protocols, and in particular the recently proposed family of AB-cast based replication schemes [28, 30, 29], represents a natural source of inspiration for the design of TM replication solutions. However, the efficiency of any transactional replication scheme is much more strained in TMs than in databases, given that the average transaction's execution time in TMs is typically several orders of magnitude smaller than in databases (in [31], for instance, we've shown that 50% of write transactions complete in less than 200μsecs when considering standard TM benchmarks). This translates into a corresponding increase of the overhead associated with the inter-replica coordination activities, urging for novel solutions aimed at minimizing such costs.

The BFC protocol aims at achieving exactly this goal, requiring just a single ABcast to commit a transaction, like in non-voting certification protocols and differently from voting ones, which incur in the costs of an additional Uniform Reliable Broadcast. On the other hand, analogously to voting certification protocols, and unlike non-voting ones, BFC avoids to flood the network with large messages carrying the whole transaction's readset. The latter aspect is particularly important given that it is well-known that the ABcast latency is significantly affected by the size of transmitted messages [18, 20] and that the transactions' readsets are frequently very large in web applications. This is also confirmed by the results of the workload characterization study of the FénixEDU application reported in Section 3.

BFC achieves such a result by exploiting the space-efficient encoding properties of Bloom Filters (BF), whose fundamentals we briefly recall in the following for the sake of clarity (the interested reader may refer to [5] for a recent survey on BF and on their applications). BFs are data structures that permit to test whether an element is a member of a set, avoiding the encoding of the whole set, but rather permitting to store a much more compact representation of it. This comes, however, at the cost of incurring in false positives (i.e., an element may

appear to be present in the set, whereas it is not), albeit false negatives are, on the other hand, not possible. More in detail, a Bloom Filter representing a set $S = \{x_1, x_2, \ldots, x_n\}$ of n elements from a universe U consists of an array of m bits, initially all set to 0. The filter uses k independent hash functions $h_1, \ldots, h_k$ with range $\{1, \ldots, m\}$, which map each element in the universe to a random number uniformly over the range. To add an element $x \in S$ to a BF, x is fed to each of the k hash functions. The array positions output by the k hash functions are then all set to 1. To determine whether an item y belongs in S, the values of the $h_i(y)$ bits are checked. If even only one of these bits is 0, it means that y is not a member of S (with no possibility of mistakes). If all $h_i(x)$ are set to 1, then x may be in S, although this may be wrong with some probability, called *false positive* probability. Interestingly, the probability of a false positive f for a single query to a Bloom Filter can be known beforehand, once the number of bits used per item m/n and the number of hash functions k are fixed, by using the following formula:

$$f = (1 - e^{-kn/m})^k \tag{1}$$

We may now start describing the BFC scheme. Similarly to existing certification-based transactional replication schemes, in BFC incoming transactions are locally processed in an optimistic fashion, avoiding any inter-replica synchronization scheme during transaction execution. Further, by leveraging on the JVSTM multi-version scheme, the BFC ensures that read-only transactions are always provided with a consistent committed snapshot. This spares them from the risk of aborts and permits to obviate the need for replica coordinations even during the commit phase. Overall, the overhead incurred in by read-only transactions due to the replication scheme is in practice almost nullified.

For what concerns update transactions, at commit time these are first locally validated to detect any local conflicts. If the local validation phase is successfully passed, the Replication Manager encodes the transaction's readset (i.e., the set of identifiers of all the objects read by the transaction) in a BF, and ABcasts it along with the transaction writeset.

As in classical non-voting certification protocols, update transactions are validated by all replicas once they are ABcast-delivered. At this stage, replicas check whether the BF of the validating transaction contains any item updated by any concurrent transactions. If no match is found, given that a BF provides no false negatives, then the transaction may be safely committed. Otherwise the transaction is aborted.

Given that the occurrence of false positives leads to the generation of unnecessary aborts, the BF size is set so to ensure that the probability p_{abort} for a false positive to induce a transaction abort is bounded by a user-tunable threshold, which we denote as *maxAbortRate*. The problem here is that p_{abort} is a function of the number of queries q that will be performed on the BF during the transaction's validation phase, but the BF has to be constructed when the transaction enters the commit phase. At this time, however, it is not possible to predict the value of q, which is determined by the number of transactions that will commit

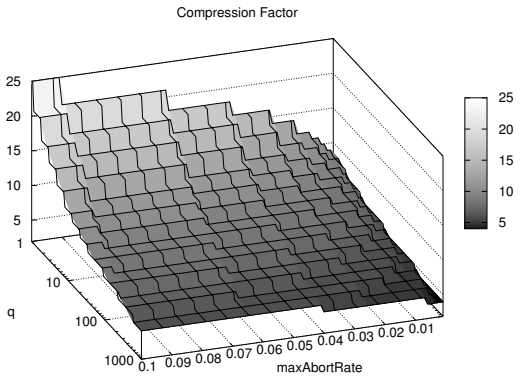


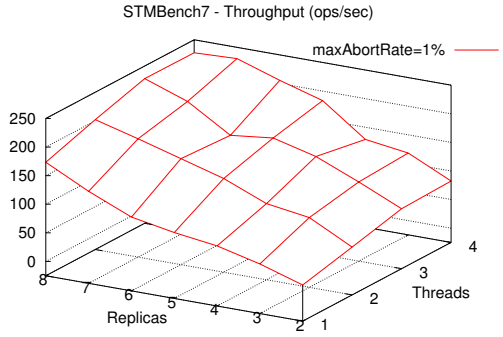
Fig. 3. Compression factor achieved by BFC considering the ISO/IEC 11578:1996 UUID encoding.

before the transaction is ABcast-delivered and by the size of the writesets of each of these transactions. Neither of these are known when the BF is created. On the other hand, it is important to highlight that any error in estimating q does not compromise consistency, but may only lead to deviations from the target *maxAbortRate* threshold. To tackle this problem, BFC uses a lightweight heuristic that estimates q through the moving average across the number of BF queries performed during the validation phase of the last c transactions to have been ABcast-delivered. Once q is estimated, we can then determine the number m of bits in the BF by considering that the false positives for any distinct query are independent and identically distributed events which generate a Bernoullian process where the probability of occurrence of a single event (namely, a false positive during a single query) is given by Equation 1. After some simple maths, we obtain the following expression for the BF's size:

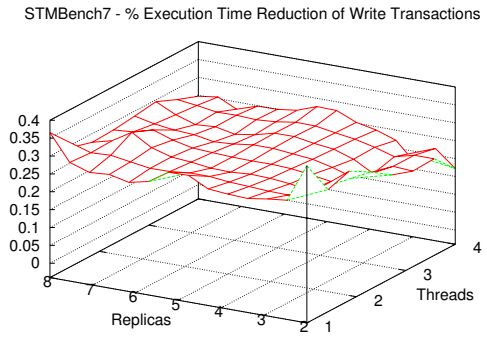
$$m = \left\lceil -n \ \frac{log_2(1-(1-maxAbortRate)^{\frac{1}{q}})}{ln\ 2} \right\rceil$$

The striking reduction of the amount of information exchanged, achievable by the BFC scheme, is clearly highlighted by the graph in Figure 3, which shows the BFC's compression factor (defined as the ratio between the number of bits for encoding a transaction's readset with the ISO/IEC 11578:1996 standard UUID encoding, and with BFC) as a function of the target *maxAbortRate* parameter and of the number q of queries performed during the validation phase. The plotted data shows that, even for marginal increases of the transaction abort probability in the range of [1%-2%], BFC achieves a [5x-12x] compression factor, and that the compression factor extends up to 25x in the case of 10% probability of transaction aborts induced by a false positive of the Bloom Filter.

To evaluate the scalability of the BFC protocol, and quantify the performance gains achievable with respect to conventional certification based protocols, we developed a prototype implementation. Differently from the architecture described



(a) Throughput



(b) % Execution Time Reduction

Fig. 4. STMBench7, read dominated with long traversals, $maxAbortRate$=1%

in Section 4, the current implementation of BFC is not yet interfaced with a persistent storage system and assumes that each replica maintains a whole copy of the DTM (hence not needing the Cache Manager component). Extending our current prototype to incorporate these modules represents an important direction for our future work. On the other hand, the current prototype permits us to evaluate empirically the performance of the BFC scheme without incurring in possible interferences generated by the coexistence of other components, such as the Persistent Storage and the Cache Manager, which address orthogonal issues with respect to BFC.

The target platform for our experimental performance study is a cluster of 8 nodes, each one equipped with an Intel QuadCore Q6600 at 2.40GHz equipped with 8 GB of RAM running Linux 2.6.27 and interconnected via a private Gigabit Ethernet. We use Appia [25, 10] as the GCS and select a classic sequencer-based implementation [19, 13] for the ABcast service.

We consider a standard, and rather complex, benchmark for TM systems, namely STMBench7 [17], which features a number of operations with different levels of complexity over an object-graph with millions of objects. Specifically, we consider the "read dominated with long traversals" configuration of the benchmark, which generates a workload closely resembling the one of the FénixEDU application (large readsets, much smaller writesets, predominance of read-only transactions). Also, we set the *maxAbortRate* parameter to the very conservative value of 1%, which should be in practice acceptable for most web applications.

In Figure 4(a) we plot the throughput (committed transactions per second) while varying the number of replicas, and the number of threads per replica. The plot shows linear speedups as the number of threads and replicas increases, highlighting the scalability of the BFC protocol. Figure 4(b) shows the performance gains achievable by BFC with respect to a classic non-voting certification scheme in terms of reduction of the execution time of write transactions, which fluctuates in the range from around 20% to around 40% and is imputable to a 3x message compression factor. This points out how the BFC scheme can achieve significant performance gains even for a negligible (i.e., 1%) additional increase of the transaction's abort rate. This makes the BFC scheme viable, in practice, even in abort-sensitive applications.

In conclusion, the BFC scheme makes it possible to use additional replicas to improve the throughput of the system and, last but not the least, permits to use (faster) non-voting certification approaches in the presence of workloads with large readsets.

6 Conclusions and future work

In this paper we have overviewed some of our recent results concerning the integration of DTMs in clusters of web application servers. In particular we have reported our experiences with the development of a complex, real web application, namely FénixEDU, which is, to the best of our knowledge, the first web application in production to rely on (D)TM technology.

We have then focused on the description of BFC, a recently introduced certification-based transactional replication scheme that permits to reduce significantly the cost of the inter-replica synchronization phase by exploiting the space-efficient encoding of Bloom Filters.

For what concerns our ongoing and future work, we are currently pursuing several orthogonal, yet complementary, research directions, which we overview in the following.

Speculative transaction execution The average latency of ABcast is, even for very small messages, in the order of at least a few milliseconds in typical data-center environments, see, e.g., [18, 20]. The completion time of (not replicated) TM transactions, on the other hand, is often two or three orders of magnitude smaller. Hence, in any ABcast based replication protocol, it is highly likely that transactions complete and stall (relatively) long before the ABcast is concluded.

This may lead to severe under-utilization of the available computing resources. Given the above considerations, we are currently pursuing the idea of speculatively exploring multiple alternative transaction serialization orders (rather than just the one suggested by the spontaneous order delivery as suggested in, e.g., [22]) so to maximize the utilization of any CPU core waiting idle for the termination of an ABcast's run.

The main challenge here is related to the fact that the number of possible serialization orders over a set composed of n elements is $n!$, which drastically reduces the probability to blindly select the correct final serialization order as the number of messages to be ordered grows. This issue raises the need for ingenious heuristics that are able to maximize the probability to drive the speculative exploration of the serialization orders towards useful trajectories.

Lease based replication mechanisms. Another orthogonal approach that we are currently pursuing is based on the idea of taking advantage from the application's data access pattern locality to reduce the frequency of activation of the ABcast-based replica synchronization schemes (and hence their inherent overhead) and to decrease the likelihood of remote conflicts.

The underlying intuition is to rely on consensus-like coordination primitives (such as the Atomic Broadcast) only to establish the ownership of a "lease" on the data items accessed by a committing transaction. On the other hand, transactions accessing data items for which the local replica already owns a lease are guaranteed not to be aborted due to a remote conflict (at least in absence of failure suspicions) and may be committed in a considerably more efficient way, avoiding the costs of ABcast-based synchronization.

By introducing the *Weak Mutual Exclusion* abstraction, see [33], we have already provided a formal specification of the lease mechanism underlying the proposed approach. At this stage, the challenge is to design and implement pragmatical, highly efficient, Weak Mutual Exclusion protocols, as well as lightweight load balancing strategies aimed at maximizing the data access pattern locality of TM applications.

Adaptive replication strategies We hypothesize that no single universal replication scheme exists that is able to effectively cope with the high heterogeneity characterizing TM workloads. Therefore, we advocate the need for developing self-adapting TM replication schemes, able to identify in a timely and automatic way the optimal replication strategy for each incoming transaction on the basis of the (estimated) size of its readset and writeset, as well as of its conflict probability.

Implementing a polymorphic replication strategy requires facing two main challenges: on one hand, ensuring the consistent interaction of different replication algorithms and, on the other hand, engineering lightweight and timely mechanisms to identify automatically the characteristics of incoming transactions [15] and the corresponding preferable replication scheme.

References

1. A.-R. Adl-Tabatabai, C. Kozyrakis, and B. Saha. Unlocking concurrency. *ACM Queue*, 4(10):24–33, 2007.
2. C. Amza, A. L. Cox, and W. Zwaenepoel. Data replication strategies for fault tolerance and availability on commodity clusters. In *Proc. of the Conference on Dependable Systems and Networks (DSN)*, pages 459–472, 2000.
3. P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
4. R. L. Bocchino, V. S. Adve, and B. L. Chamberlain. Software transactional memory for large scale clusters. In *Proc. of the Symposium on Principles and Practice of Parallel Programming (PPOPP)*, pages 247–258. ACM, 2008.
5. A. Broder and M. Mitzenmacher. Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4):485–509, 2003.
6. J. Cachopo and A. Rito-Silva. Combining software transactional memory with a domain modeling language to simplify web application development. In *Prof. of the International Conference on Web Engineering (ICWE)*, pages 297–304, 2006.
7. J. Cachopo and A. Rito-Silva. Versioned boxes as the basis for memory transactions. *Sci. Comput. Program.*, 63(2):172–185, 2006.
8. V. Cardellini, E. Casalicchio, M. Colajanni, and P. S. Yu. The state of the art in locally distributed web-server systems. *ACM Comput. Surv.*, 34(2):263–311, 2002.
9. N. Carvalho, J. Cachopo, L. Rodrigues, and A. Rito Silva. Versioned Transactional Shared Memory for the FenixEDU Web Application. In *Proc. of the Workshop on Dependable Distributed Data Management (WDDDM)*. ACM, 2008.
10. N. Carvalho, J. Pereira, and L. Rodrigues. Towards a generic group communication service. In *Proc. of the International Symposium on Distributed Objects and Applications (DOA)*, 2006.
11. R. G. G. Cattell, D. K. Barry, M. Berler, J. Eastman, D. Jordan, C. Russell, O. Schadow, T. Stanienda, and F. Velez, editors. *The Object Data Standard – ODMG 3.0*. Morgan Kaufmann Publishers, Inc., Los Altos, USA, 2000.
12. M. Couceiro, P. Romano, N. Carvalho, and L. Rodrigues. D^{2}STM: Dependable Distributed Software Transactional Memory. In *Proc. of the Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE Computer Society Press, 2009.
13. X. Defago, A. Schiper, and P. Urban. Total order broadcast and multicast algorithms: Taxonomy and survey. *ACM Computing Surveys*, 36(4):372–421, 2004.
14. S. Frølund and R. Guerraoui. e-Transactions: End-to-end reliability for three-tier architectures. *IEEE Transaction on Software Engineering*, 28(4):378–395, 2002.
15. S. Garbatov, J. Cachopo, and J. Pereira. Data access pattern analysis based on bayesian updating. In *Proc. of the 1st Simpósio de Informática (INForum)*, Lisbon, Portugal, Sept. 2009.
16. J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. of the Conference on the Management of Data (SIGMOD)*, pages 173–182. ACM, 1996.
17. R. Guerraoui, M. Kapalka, and J. Vitek. STMBench7: a benchmark for software transactional memory. *SIGOPS Oper. Syst. Rev.*, 41(3):315–324, 2007.
18. R. Guerraoui, R. R. Levy, B. Pochon, and V. Quema. High throughput total order broadcast for cluster environments. In *Proc. of the International Conference on Dependable Systems and Networks*, pages 549–557. IEEE Computer Society, 2006.
19. R. Guerraoui and L. Rodrigues. *Introduction to Reliable Distributed Programming*. Springer, 2006.

20. M. Kaashoek and A. Tanenbaum. An evaluation of the Amoeba group communication system. In *Proce. of the International Conference on Distributed Computing Systems (ICDCS)*, page 436. IEEE Computer Society, 1996.
21. P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In *Proc. of the International Symposium on Computer Architecture (ISCA)*, pages 13–21. ACM, 1992.
22. B. Kemme, F. Pedone, G. Alonso, and A. Schiper. Processing transactions over optimistic atomic broadcast protocols. In *Proc. of the International Conference on Distributed Computing Systems (ICDCS)*, page 424. IEEE Computer Society, 1999.
23. C. Kotselidis, M. Ansari, K. Jarvis, M. Lujan, C. Kirkham, and I. Watson. DiSTM: A software transactional memory framework for clusters. In *Proc. of the International Conference on Parallel Processing (ICPP)*, pages 51–58, 2008.
24. K. Li and P. Hudak. Memory coherence in shared virtual memory systems. In *Proc. of the Symposium on Principles of Distributed Computing (PODC)*, pages 229–239. ACM, 1986.
25. H. Miranda, A. Pinto, and L. Rodrigues. Appia, a flexible protocol kernel supporting multiple coordinated channels. In *Proc. International Conference on Distributed Computing Systems (ICDCS)*, pages 707–710. IEEE, 2001.
26. OJB. Object/Relational Bridge - OJB, http://db.apache.org/ojb, 2007.
27. M. Patino-Martínez, R. Jiménez-Peris, B. Kemme, and G. Alonso. Scalable replication in database clusters. In *Proc. of the International Conference on Distributed Computing (DISC)*, pages 315–329. Springer-Verlag, 2000.
28. F. Pedone, R. Guerraoui, and A. Schiper. The database state machine approach. *Distributed and Parallel Databases*, 14(1):71–98, 2003.
29. F. Perez-Sorrosal, M. Patino-Martinez, R. Jimenez-Peris, and B. Kemme. Consistent and scalable cache replication for multi-tier J2EE applications. In *Proc. of the International Conference on Middleware (Middleware)*, pages 328–347. Springer-Verlag, 2007.
30. L. Rodrigues, H. Miranda, R. Almeida, J. Martins, and P. Vicente. The GlobData fault-tolerant replicated distributed object database. In *Proc. of the First EurAsian Conference on Information and Communication Technology*, pages 426–433. Springer-Verlag, 2002.
31. P. Romano, N. Carvalho, and L. Rodrigues. Towards distributed software transactional memory systems. In *Proc. of the Workshop on Large-Scale Distributed Systems and Middleware (LADIS)*, 2008.
32. P. Romano, F. Quaglia, and B. Ciciani. A lightweight and scalable e-Transaction protocol for three-tier systems with centralized back-end database. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1578–1583, 2005.
33. P. Romano, L. Rodrigues, and N. Carvalho. The weak mutual exclusion problem. In *Proc. 23rd IEEE International Parallel and Distributed Processing Symposium*. IEEE Computer Society Press, to appear.
34. M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *Proc. of the 33rd international conference on Very Large Data Bases (VLDB)*, pages 1150–1160. VLDB Endowment, 2007.
35. Terracotta Inc. Terracotta. http://www.terracotta.org/.
36. Transaction Processing Performance Council. *TPC Benchmark^{TM} W, Standard Specification, Version 1.8*. Transaction Processing Perfomance Council, 2002.
37. Transaction Processing Performance Council. *TPC Benchmark^{TM} TPC-APP, Standard Specification, Version 1.0*. Transaction Processing Perfomance Council, 2004.